Math

Ie6r
o.956 Report No. UIUCDCS-R-79-956

op.2

UILU-ENG 79 1707

USER'S GUIDE TO EUREKA AND EURUP

by

Thomas G. Burket and Perry A. Emrath

February 1979

NSF-OCA-MCS77-27910-000039

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-79-956

USER'S GUIDE TO EUREKA AND EURUP

by

Thomas G. Burket
Perry A. Emrath

February 1979

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Supported in part by the National Science Foundation under Grant No. US NSF MCS77-27910.

TABLE OF CONTENTS

INTRODUCTION

EUREKA is an experimental system designed for research in the field of information retrieval. Part 1 of this report describes the current state of EUREKA and how to use it to conduct searches of a database. It is divided into four sections - basic features, query language description, general comments about EUREKA use and a glossary of terms and rules.

EURUP, for EUREKA UPDATE, is the system which builds and maintains databases for use by EUREKA. Part 2 discusses the logical organization of a database, as well as how to construct or modify the component files. It has five sections - introduction, database structure, EURUP input and output formats, EURUP functions and command language, and general comments about efficient EURUP use. Both EURUP and EUREKA run on a PDP 11/40 minicomputer.

PART 1 - EUREKA

1. Basic EUREKA features

As a package, EUREKA is unique, although it shares many features with other existing retrieval systems. The major points of its structure are:

1) It is interactive.
2) The full text of each document is available.
3) The index contains every word that appears in the documents (instead of a controlled vocabulary).
4) A document is divided into several "contexts", such as author, title and sentence, and searches may be confined to particular contexts.
5) Results of queries are stored for later use.
6) A user-defined thesaurus may be used.
7) Several databases are potentially available.

Before giving more details, let's examine a typical user's approach to EUREKA. She has a particular area to investigate, presumably the type that can be handled by reading the text of relevant documents. EUREKA is not a question-answer system where someone might ask, "How many cities in Illinois have over 100,000 people?", and receive the answer "3". Instead, the user must formulate her request as a set of terms that EUREKA can find within the text, such as "city", "population", "residents" and so on.

Armed with her request, she "signs on" to the system and the correct database and enters a search command. EUREKA responds with statistics on the number of documents containing the given terms and (we hope) the desired information within them. She probably would then scan and evaluate some of the documents.

After this reading, the user has three choices:

1) The question has been answered so exit the system or go on to another query.
2) Finding the answer appears hopeless so give up.
3) The result is not quite right but the answer appears within reach so modify the original query for a more accurate search.

These steps can become very complex and require the subsidiary features of EUREKA. We will begin the details by briefly describing the system command formats and functions. The glossary at the end of the manual defines various crucial terms appearing from now on.

2. EUREKA Query Language

2.0 Command Summary

bye - disconnects the user from EUREKA
change - changes the name of a query set or changes a thesaurus entry
comment - attaches a comment to an existing query set
db - allows the user to access a different database
delete - removes a query set, comments or a thesaurus entry
enter - defines a thesaurus entry
find - does searches of the database for terms in some context
freq - allows scanning the database index to see what terms exist and to see
    statistics about them
guide - enters the lesson facilities
help - displays the key ideas about EUREKA commands or some other area to help
    the user who is confused or has forgotten some details
lp - causes certain output to go to a line printer instead of the terminal
make - forms a new query set out of some combination of existing sets and
    documents
message - leaves a comment or message for "The Management"
print - displays document text, query set or thesaurus information
t - sets flags for thesaurus use
term - like freq only without statistics
users - displays a list of signed-on EUREKA users
words - displays keywords from a document set

Detailed command descriptions follow. Keep in mind that EUREKA is not static and that the formats may change. The help files (see help command) are maintained as up to date as possible so that information about changes is available.

Note on notation: when defining a command format, the notation <word or phrase> means the user must enter something of that type in that location. The notation distinguishes command parameters from the keywords that must appear.

2.1 Signing on and off EUREKA

Naturally, the first step in using EUREKA is to get connected to it and the desired database. When EUREKA is ready for someone, it types on the terminal:

```
Please type your name:
```

and then waits for someone to sign on.

Prior to the first encounter, a person should be assigned a unique user name, possibly matching his real name. When given a user name, each person is also given a default database that he is automatically connected to at signon. Getting on then is as simple as typing that user name and hitting the return key. To override the default database, the user name may be followed by a comma and some other database name. If EUREKA responds with a line similar to "Query #1" then he is on the system. However, if a message like "Database not found . . ." appears, then either that data is not available or he may have just mistyped the database name. In that case, the user is connected to the system as a whole, but not to any data. Most commands will not work without first entering a db request (see Section 2.6) to perform the data attachment. Examples:

```
Please type your name: tom          (user "tom" is logged on
Query #7                             to his default database)
#

Please type your name: tom,west      ("tom" is logged on
Query #7                              to the database
#                                     named "west")
```

The simplest command of all is BYE (or BY), which disconnects the user from EUREKA. Anytime the user is shown a "#", as above following the Query #7, typing "bye" signs him off and has EUREKA ask for another person. For example,

```
Please type your name: tom
Query #1
#bye
    (a message appears about how long "tom" was on system)

Please type your name:    (ready for another user)
```

2.2 Find command

The find command is the most important one in EUREKA, as it causes the database to be searched. It can get quite complex, so we will start with its basic form of

```
find <expression>        or        f <expression>
```

where <expression> is a combination of terms.

For example, "f tax" means search the entire database for documents containing the word "tax". This has limited power, so combinations of words are permitted, as in

```
f income or tax
f income and tax
f tax and income or sales
f tax and (income or sales).
```

The first one means find documents that have either "tax" or "income" or both, while the second requires that both "income" and "tax" must be in the document for it to be retrieved by EUREKA.

The third example shows that these operators (AND/OR) may appear together in the same expression, but an ambiguity does arise. In EUREKA, the request would be interpreted as

- find all documents that contain "tax" and "income" together, or any document having "sales"

instead of

- find all documents that contain "tax" and either "income" or "sales".

Whenever ANDs and ORs appear, the AND is always evaluated first because of its higher precedence. If the user had wanted the other meaning in the example, then the fourth case above would provide that request. Parentheses allow grouping of terms to override the natural or default usage. This is analogous to algebraic expressions where A+B*C might mean either (A+B)*C or A+(B*C).

Abbreviations do exist for both operators, "*" and "&" for AND, "+" for OR, so "f a and b or c" could be entered as "f a & b+c". The symbols are completely equivalent to the words, except they need not be surrounded by spaces as the words do. That is "f income and tax" could not be typed as "f incomeandtax", but "f income&tax" is acceptable.

Notice that "f income and tax" will select a document that has "income" anywhere in it and "tax" also somewhere within. What if we wanted to find "income tax", where the words are next to each other? EUREKA permits the grouping of several words together into one term by surrounding them by single quotes. For example,

f 'income tax'   would solve our problem, and similarly,

f 'income tax' or 'internal revenue service'.

What if we want to find tax, taxes, taxpayer and so on? There is no need to type all the possible words. By putting a "?" at the end, as in "f tax?", all words that start with "tax" are searched for. However, if <u>all</u> the endings aren't desired, such as "taxicab" or "taxation", then we must use a list of the acceptable endings. For example, "f tax or taxes or taxpayer".

The discussion of term and thesaurus commands gives ways to help handle this problem (Sections 2.9 and 2.18).

Just as suffixing is allowed, putting the question mark as the first character asks EUREKA to find all words with the given ending. Thus, "f ?ation" could produce "taxation", "representation", "sensation", etc. EUREKA checks the entire index in this case, so prefixing is relatively slow and is not recommended. To help prevent careless use, the ending must be at least three characters. "?y" fails, but "?ogy" passes.

A "?" may appear at both ends of the same word. "?ize?" may not be very useful, but it would match terms "socialize" and "computerized". Prefixing and suffixing work for quoted phrases, too, as with "income tax?" looking for "income taxes" and other similar strings. However, a "?" inside a term is treated as an actual question mark. The exact rules for what constitutes a term appear in the glossary under "term".

The results displayed after a search request are a compact representation of the documents retrieved. For example, this is a sample set.

```
f mathematics
66 documents responded to index search
66 documents are in this set
The 20 documents with the most occurrences are:
    1(  2)    5(  3)   10(  1)   42(  1)  229(  4)
  252(  1)  428(  1)  527(  1)  703(  5)  710(  4)
  715(  2)  776(  2)  780(  9)  825(  2)  837(  2)
 1107(  3) 1168(  4) 1412(  2) 1723(  2) 1774(  7)
```

The search found 66 documents containing "mathematics" somewhere in the text. The table of numbers gives those documents with the most frequent use of the term. The first number in each pair is the document number and the second i: the number of occurrences for the term. Thus document 229 had "mathematics" in it four times, and 1723 had it twice.

The list is sorted in document number order, except that only the top 20 matches are listed. We see that document 780 had nine occurrences, 1774 had seven and so on down to several with just one. By this ordering, we know that n other documents could have had more than one occurrence, for otherwise it would have been in the table. When many documents have the same lowest value, the one with the lower document numbers are given. In this example 66-15=51 documents must have had just one occurrence of "mathematics", but only those five with th smaller document numbers appear.

As another example, consider

```
f 'computer architecture'
44 documents responded to index search
Do you wish to abort this query? n
Do you wish to abort this query? n
44 documents required full-text search
```

```
12 documents are in this set
   4(  1)  271(  2)  651(  1) 1650(  1) 1671(  1)
 1708(  7) 1713(  1) 1722(  3) 1764(  1) 1839(  2)
 1877(  1) 1892(  1)
```

Several new ideas show up in this request for a phrase. EUREKA checked the index and found that 44 documents had both "computer" and "architecture" in them. However, the user asked for "computer architecture", forcing the words together and in that order. This forces EUREKA to do a "full-text search", scanning all 44 documents to see if that requirement is met. In this example, 12 of the 44 are retrieved and the other 32 do not have the exact pair "computer architecture".

The inquiry about aborting the query allows the user to exit a request that may be time-consuming. Full-text searching is relatively slow because of the considerable text processing needed, and the user may not want to wait for its completion. This is especially handy if many documents are retrieved. EUREKA prompts the user after every 16 documents, so the above queries occurred after 16 and then 32 different articles had been scanned.

2.2.1 _Find contexts - the "in" clause_

The above discussion on the find command dealt only with its basic form. The command does have several options good for a substantial increase in power. One of these is the "in" clause, or context option, which appears as

```
f <expression> in <context list>.
```

The context clause tells EUREKA to confine its searching to only those specified regions of the document. A context is a predefined division of a document, such as author, title, body and references. Each document in a database has at least some subset of the contexts contained within it. Each document within the database has the same contexts defined, but those specific names depend on the database. Only sentence and paragraph are common to every database.

The context names may be abbreviated, the actual shortening permitted again being dependent on the database. The usual limit is one or two characters, with two required whenever some other context name begins with the same letter and has been given a higher priority. For example, "sentence" abbreviates as "s", so if a context "source" existed, its shortest form would be "so".

A context list is one or more context names separated by commas. A comma implies OR, so "in author,title" means find it in either the author or the title section, or both.

Expressions with "in" clauses can get arbitrarily complicated, since "in" may appear many times. We'll start with cases of only one clause. The two key ideas to remember are that the clause is actually part of the expression and that it applies to everything to the left (up to the left end of its parenthesis level). Examples are

```
f hemingway or frost in author
f (hemingway in au) or (frost in au)
f hemingway in author or frost
f hemingway or (frost in author)
```

The first case asks for those documents in which either "hemingway" or "frost" is the author, ignoring ones where either name appears elsewhere in the document, such as in a paragraph discussing the virtues of Frost's poems. Of course, if a document has "hemingway" in the author section and also somewhere else, it will still be retrieved. Note that "in author" affects everything to the left.

In the second case, we have exactly the same interpretation. The parentheses around the "frost" term cuts off the rightmost "in au", requiring an additional clause for "hemingway". However, the parentheses around the "hemingway" clause are redundant, since nothing is to the left of it.

With "in author" moved to the left term in the third example, the request means find documents with "hemingway" in the author section or "frost" anywhere. The fourth example again demonstrates context cutoff by a parenthesis. "Frost" must appear in the author region, but "hemingway" may be anywhere, so this is the opposite of number three.

To search for "hemingway" in either the author or the title, try

```
f hemingway in author,title
```

but to concentrate only on the most famous Hemingway, use

```
f ernest and hemingway in au,ti.
```

In the example

```
f (tax or taxes) and income in sentence
```

the "in sentence" clause applies to the entire expression. The parentheses do not affect the context, since the "in" is not <u>inside</u> the parentheses. It "invades" parentheses to the left.

Stepping up to two clauses brings in considerable power as well as added complexity. The simplest case is two in a row, as with

```
f hemingway or frost in author in title
```

which is the same as "... in author,title". A more typical example and its effective interpretation is

```
f hemingway in au or frost in title
f (hemingway in au,ti) or (frost in ti).
```

The "title" affects both its neighboring term and all else to the left, thus combining with "author". This example is also ILLEGAL, breaking a conflict rule described later. The interpretation and the illegality is probably <u>not</u> what the reader expected!

To get independent operands, parentheses are needed. Continuing with the literary example, we might want to find all documents with "hemingway" in the title and "baker" as the author. The dual "in" clause allows this request to be handled in one try with

```
          f (hemingway in title) and (baker in author).
```

The parentheses confine the context to the desired term.

More examples are

```
          f (income & tax in s) and (illinois and state in p)
          f 'cook county' and (prison or jail) in sentence
          f (1976 and may in date) and (database or 'data base' in ti)
          f (knuth in au)+(cacm in source)&(1978 in da).
```

An "in" may appear after every term in a search expression, although such use tends to be confusing or cause errors with misplaced parentheses.

The user might think that he can sprinkle "in" clauses without restrictions on meanings. Consider

```
          f (a & b in p) and c in s.
```

This should be interpreted as "((a&b in p,s) and (c in s))". However, "p" and "s" conflict, since sentences are subsets of paragraphs. The attempt to put a tighter constraint at an outer level is illegal. The reverse case of "(a & b in s) & c in p" is ok, and the more general paragraph restriction will apply at only the outer level, as was probably expected. That is, "a" and "b" must still be in the same sentence, plus "c" must exist somewhere in the same paragraph.

The other main error comes from inconsistency in section names (author,title,date,etc., but not s or p). A specified name must agree exactly with any outer level context lists affecting it and having a name in common. For example, these two cases fail:

```
          f a in au,ti and b and c in au
          f a in au and b and c in au,ti.
```

The reason is that the value of "a" is modified by both "au" and "au,ti". Which is meant?

The rule is this: if a term is modified by an "in" clause and then another "in" at an outer level, then the contexts at each level must agree <u>exactly</u>. The case of "hemingway in author and frost in title" fits this rule. "Hemingway" is affected by an inner level (author) and an outer context (title), but since the two don't match - error! Any differences in contexts must be confined to parenthesized terms, such as "hemingway in au or (frost in ti)", which cuts off the influence of "ti". An alternative is "hemingway in au,ti or (frost in ti)", which cuts off the "ti" again and makes the combination of "au,ti" explicit and thus legal.

One note: this specific matching rule does not apply to sentence or paragraph, so the matching check is done after excluding any s or p specifiers.

The most common use of contexts comes through the sentence and paragraph divisions. Many requests want a group of words in the same general area, such as only a few words apart. EUREKA does not have the capability to find two words, say, within three words of each other, but restricting a search to the same sentence or paragraph is a powerful approximation. Actually, this method is often better than a distance measurement because of large variations in English grammar and sentence structures across writing styles and document contents.

2.2.2 Search sets - the "from" clause

Another useful option for the find command is the "from" clause, which specifies what set of documents EUREKA should search. When no "from" clause exists, the entire database is examined. Its most fruitful use is to narrow the results of a previous query to a smaller set of documents. If the user knows the desired answer is somewhere within a set, she may request that searching be confined only to that set, thus producing a better result in less time. Its basic form is:

```
        f <expression> from <set expression>
```

or if the "in" clause is also used,

```
        f <expression> in <context> from <set expression>.
```

The <set expression> specifies a combination of existing query sets and documents to use as the search set. As with <expression>, the set expression is a series of terms separated by operators. For sets, these are

```
1) and, &, *   :  e.g. 2 and 3.
2) or, +       :  e.g. 2 + 3.
3) minus, -    :  e.g. 2 - 3.
```

AND and OR carry their original meanings here, with the addition of MINUS. This new operator means "but not", as in "documents in query set 2 but not those which are also in set 3".

Each term in the set expression must be one of

1) Query set number.
2) Query set name.
3) List of document numbers.

All three are converted into a list of documents and subjected to some combination with the lists from other terms. Some examples are

```
f tax from last
f tax from irs
f tax from 3
f tax from [1,2,3,4]
f tax from irs-3
f tax from irs+3-[10,11,22].
```

The first case is the most common use of "from", where the search is conducted only over those documents retrieved in the most recent query. "Last" is a special keyword known to EUREKA. The second example assumes the user had given the name "irs" to an existing set, causing its documents to be fetched for searching.

Each set with a name still has a query number assigned at the time of the find command. If "irs" were the name of query set number three, the two successive examples would be equivalent.

A list of numbers surrounded by square brackets is always interpreted by EUREKA as a list of specific documents. In the fourth example, searching would be confined to documents one through four in the database.

The fifth case would search documents from set "irs", except those also in query #3. If "irs" and three were the same, the search set would be empty and nothing would result.

The more complex final example demonstrates all three types of set forms, generating a set containing documents in either "irs" or #3, but removing documents 10, 11 and 22 if they are in either of those two sets.

When the "in" clause is used, the "from" must appear after the context if it is to be used. Note that "from all" is the same as no "from" at all, and that parentheses may not be used. See the make command (Section 2.13) for techniques in constructing complex set expressions.

2.2.3 Naming sets using "="

Remembering query numbers can be difficult, so EUREKA allows labels on queries. One way to attach a name is to do it when the set is formed with the find command. The form needed is

```
f <expression> = <name>        or in full form,
f <expr> in <context> from <set expr> = <name>.
```

The name supplied

1) Must not already exist.
2) Must start with a letter.
3) Can be up to ten letters and/or digits.
4) Cannot be a keyword, like "last".

For example,

```
Query #3
#f tax = irs
```

searches for "tax" in the whole database and forms query set #3 out of the retrieved documents. The label "irs" is attached to the set, so "f blah from irs" would be valid as long as the set and the name existed.

2.2.4 Commenting sets

Our final embellishment permits commenting the query set. A string of characters surrounded by double quotes (" and not '') and appearing at the end of the command defines a comment. This note to oneself may be viewed later and could serve as a reminder of some point about the query. For example,

```
f evasion from irs "tax evasion penalties"
```

does a search of the "irs" set, creates a new query set and includes the quoted string as a comment.

2.2.5 Find command summary

An example of a full form find command may now be given.

```
f tax and income in sent from last = taxation "income tax laws"
```

has all the optional clauses, and ordered properly. Anything beyond "income" is optional, but, if used, must follow the strict rules. Most of the options reappear in other commands, such as DELETE and CHANGE, where more details will be explained. Browsing the documents retrieved by FIND is the function of PRINT, which will be discussed next.

Following the search, EUREKA stores the search expression and the documents retrieved. The expression saved is the effective one actually used to perform the search, although without thesaurus expansion. Thus, for the sequence

```
f a          and then
f b from last
```

the effective second request is "f b and a". "B & A" would be stored as the search expression for that query. In general, any time a "from" clause appears, the net expression is "<new expression> and <expression from existing set>".

Theoretically, the fully expanded thesaurus terms should be saved instead of just the user-typed terms, but the amount of storage required is prohibitive. Therefore, to get consistent results when printing from a set, the thesaurus must be in the same state as when the search was done.

2.3 Print command

PRINT displays document text, query sets, and thesaurus entries. See Section 2.18.3 for thesaurus information. Printing query sets is simplest, so we'll begin there.

2.3.1 Print query

This command form recalls data about previous queries, the data being:

1) Query number and name, if any.
2) The search expression used for generating the query (without thesaurus expansion).
3) The number of documents in set.
4) All document-frequency pairs (not just the top 20 as with FIND).
5) Any existing comments.

The command format is one of:

```
print query <number or name>
print query <# or name> to <# or name>
print query <# or name>,<# or name>,...,<# or name>
```

with "p q" a valid abbreviation for "print query".

The first fetches and displays the single query specified, while the second displays a series of them. For example,

```
p q 3
```

would show query set #3, but

```
p q 3 to 10
```

would display #3 and go on through #4,#5 and up through #10.

To print a group of queries not in the same range, a list is given. Then only those specified query sets appear. Two examples are

```
p q 1,4,last
p q irs,6
```

EUREKA allows interaction with the display in many cases, depending on the amount of data being shown. Printing is stopped every 20 lines, if necessary, and the user is prompted with a "*". She then may

```
1) Hit <return> to continue.
2) Type "k" to kill the printing.
3) Type "h" to get help.
4) Type "d" to stop printing and delete this set.
```

If the command includes the "to" clause or a set list, printing also pauses between queries. The same four options apply there, too. Two alternatives exist to typing "to". A dash ("-") or double dots ("..") between the range bounds is treated just as "to". More examples are

```
p q irs .. last
print q 1-50
p q 27 to 300.
```

The final case could represent a mistyping. The user need not worry about causing too much output by having the system go all the way to 300 instead of the intended 30. She may always exit by typing "k" at any prompt between queries.

2.3.2 Print context

EUREKA's other print command retrieves and displays text from documents. Its form is

```
print <context list> from <set>.
```

The contexts permitted here match those of FIND, with a list of contexts separated by commas acceptable. The set rules, however, are more restrictive by not allowing an expression. It must be a single query set number or name, or a document list in square brackets.

Both the context list and the "from" clause may be left off. If there is no "from", then "from last" is assumed, while if no context is given, "document" is the assumed value. Therefore, "p", "p document", "p from last" and "p document from last" all have the same meaning.

If <set> is a query set and not just a document list, EUREKA displays the contexts from each document in the set, flagging those lines that contain one of the search terms from the original query. For example, consider this sequence:

```
Query #3
#f tax
...
Query #4
#p sent from last
```

EUREKA would begin showing sentences from the documents retrieved by query three. The only sentences would be those having "tax" somewhere in them. The first sentences would be from the document with the most occurrences of "tax". When that document is finished, EUREKA would continue with the document containing the second most occurrences of "tax", and so on to the last document in the set.

The specific rules for what portions of a document are displayed get complex, so they will only be outlined here. The most important condition is whether the print context list contains "sentence" or "paragraph" ("s" or "p"). If it does _not_, then the specified contexts are displayed, with any line flagged if it has one of the search terms. Any context specifiers from the FIND are ignored. The FIND restricts the document selection process only and does not affect what happens when printing. For example, given

```
f money in title
p body
```

the entire body of each document is shown, with any occurrence of "money" marked. Titles are not displayed.

Assume the print context does contain "s". The discussion also applies similarly for "p". Then any additional contexts are first displayed, as above, with any search terms flagged. After that, a second pass is made over the text, looking for selected sentences to show. If no "s" appeared in the search expression, then any sentences with an occurrence of a search term are displayed, even if they already appeared during the first pass. The sentences must be within the contexts of the search expression, however. If "sentence" was in the search expression, the above applies except a sentence shown on the second pass must have an instance of the entire expression, not just one of the terms. As an example,

```
f digital and computer in body
p s,title
```

prints the titles, flagging those lines that happen to have "digital" or "computer". It then displays any sentences from the body that have either term.

As an alternative,

```
f digital and computer in body,s
p title,s
```

also prints titles, but then shows only sentences from the body with both "digital" _and_ "computer".

The number of different cases gets large, and they are not always handled in the obvious way, so it is best to experiment and keep in mind this idea: with FIND, use "in" to restrict the search to specific contexts, while use contexts with PRINT to show specific parts of the documents in a query set, regardless of the search expression.

2.3.3 _Browse mode_

The user may break the pattern of simple text display and interact with EUREKA during the display by using "browse mode". Some of the capabilities are skipping to the next document, viewing other contexts within a document, and killing the whole display process.

Browsing is implemented by remembering where the user was in the main print stream before entering browse mode. So, if the above example of printing sentences with "tax" were interrupted by browsing, exiting browse mode would return EUREKA to printing more "tax" sentences.

Whenever a context is completed, or after 20 lines, EUREKA prompts the user. He may continue the current state by hitting <return> or using one of the browse commands. They are:

p - print next paragraph

s - print next sentence

5p - skip to the fifth paragraph from the current one. The general form is np, where n is some number. -5p means back up five paragraphs. The same format works for sentences, such as -4s and so on. If the end of the context is reached (like the end of body), EUREKA will not advance without a context request (see below).

skip - go to the next document in the set

kill - stop the entire printing process

end - leave browse mode and continue with original printing

help - get help on browsing

<context> - by typing any one of the contexts legal in the database, that part of the document is entered and display is begun. For example, typing "references" would move to the references division (if it exists). While in that section, any "p" or "s" browse request stays confined to the references. Leaving the section is possible only by entering another context or using one of "skip", "kill" or "end".

The current mode (print vs. browse) is always identified at the top of the output screen. Browse mode begins whenever a context name (including p,s) is input at a prompt.

Browse mode can get very confusing! We recommend extensive experimenting to get the feel of its capabilities and functions. Many useful tools for browsing do not exist, such as looking for a particular term while browsing or skipping back to an earlier document. We hope to add some of these features in the future.

2.4 Change command

CHANGE assigns a name to a query set or changes an existing name. The name may then appear in FIND "from" clauses, PRINT requests, and so on, in place of the query number. The command form is

```
change <query # or old name> to <new name>
```

The new name must match the rules specified in the FIND description and in the glossary under query set name. Examples are

```
change 3 to taxation

change prisons to jails
```

Assuming query #3 had no name, the first command assigns "taxation" as its name. If it did have a name, that label would be deleted. In the second case, "jails" replaces "prisons" as the set name.

Change also has a version for modification of synonyms, entered as "change syn <name>". The thesaurus section describes this command.

2.5 Comment command

The comment command attaches comments to a query set. This is similar to the "comments" clause of FIND. The form needed is

```
comment <set # or name> "some string"
```

as in

```
comment 3 "income tax rules"
com jails "joliet state prison escapes".
```

The comments are displayed during PRINT QUERY requests. If the set already has comments, the new string is added with the existing comments preserved.

2.6 DB command

A user wishing to switch to another database may avoid signing off and then signing on to the other one by using the db command. This simple statement is just "db <database name>". For example, if a user were in database "west" and wanted to use database "cr" for a while, he could follow this sequence:

```
Query #10
#db cr

Query #10
#   (commands while in "cr")
...
Query #17
#db west      (back to original database "west")
```

If a bad name is given, the user is left attached to no database and must either try again with a new, accurate name or else exit EUREKA with BYE.

Section 2.1 mentioned a case where DB must appear. Consider this:

```
Please type your name: tom,xxx
****************
DATABASE UNAVAILABLE OR ERROR IN NAME

Query #12
#f something
****************
You must pick a database (use DB command)

Query #12
#db west
```

In this example, no database "xxx" exists, so "tom" was left without searchable data. Giving the db command with "west" solved the problem and completed the signon process (assuming, of course, that "west" is a valid database).

2.7 Delete command

DELETE has four forms:

```
delete query <set list>
delete comments <set list>
delete query all except <set list>
delete syn <synonym list>
```

The first form removes query sets from the user's collection. A query that no longer has value should be removed to keep from cluttering up the user's space. The set list in this command is a series of query numbers and/or names separated by commas, or a range specification using "<set> to <set>". Any query names and comments are also purged with the set. Examples are

```
delete query last
delete q 3
del q 3,5,11,jails,irs
del q 1,2,3,4
del q 1..4       (these 2 are the same)
del q jails to irs
del q jails-irs  (these 2 are the same)
```

Note the format is just as with PRINT QUERY.

DELETE COMMENTS only removes the comments from the set(s) specified. All query set information survives. For example,

```
delete comments last
del com 3,5,jails
del com 10-20
```

One simplification for deleting everything is available - the "delete q all" and "delete query all except <set list>" phrases. The first removes all query sets, comments, and names, and returns the user to query #1. If a few queries should be preserved, then "except" is used, with either a list or range. Thus,

```
del query all except 5,jails
```

deletes every query except #5 and the one named "jails".

```
del comments all
```

also affects every query, but just clears out all the comments from them. Again, the sets survive.

The keyword "query" is actually optional in all the above examples. Its use is encouraged for consistency with PRINT QUERY.

DELETE SYN purges synonyms from the user's thesaurus. See Section 2.18.

2.8 <u>Enter command</u> - see thesaurus, Section 2.18

2.9 <u>Freq and term commands</u>

These two commands are closely related. They provide access to the actual database index and show the exact words available for search terms.

Each command has one parameter - a string used to compare against terms in the index. Any word that begins with the same series of letters as the string is shown. The display pauses every 20 lines, if necessary, and the user may then hit <return> to continue or type "k" to quit. An example from a possible database is

```
Query #6
#term tax
tax
taxable
taxation
taxed
taxes
taxi
taxicab
taxpayer
taxpayers
```

Notice that each word starts with "tax". What if you want to match on the end of a word? EUREKA provides prefixing also, using a "?" as with search terms. For example, "?able" would list terms like "sociable" and "taxable".

FREQ outputs more detailed information about the terms. For each one, the number of documents in which the word appears is given, along with its total frequency of occurrence. A third number, used by the system people, is to be ignored. The FREQ form of the above example is

```
Query #10
#freq tax
tax         33     313     017306
taxable     11      45     017453
taxation     3       3     017620
taxed        4       6     017765
taxes       19      95     020132
taxi         1       1     020277
taxicab      1       1     020444
```

```
        taxpayer   17     166     020611
        taxpayers   5      38     020756
```

We see that "taxpayer" occurs in 17 documents a total of 166 times. This would match the result found by a "find taxpayer" command. FREQ permits prefixing, as TERM does.

2.10 Guide command

EUREKA has a feature that provides lessons on particular subjects, such as how to use the system. The user is stepped through the subject matter and any exercises provided by the developer of the lesson. Eventually he returns to the normal EUREKA environment.

When a lesson is available, typing "guide" puts the person into the lesson programs. Currently, GUIDE needs the "cai" database, so EUREKA switches the user to "cai" and back to the original database on exit from GUIDE. If CAI is not available, GUIDE cannot be entered.

2.11 Help command

Sometimes a user forgets a command format or a detail about how the command works. For times like these, EUREKA offers a help feature. Whenever the user is prompted for a new query by the "#", help may be requested simply by typing "help" or "h". The system then displays some general help information and a list of topics.

After scanning this list the user selects the topic of interest, types the name, and then reads what EUREKA has to say. When the display is finished, EUREKA goes back to prompting for another query.

The help-getting process may be shortened by following "help" with the name of a topic. If the name is legal, then EUREKA bypasses the general step and goes right to the requested data. For example,

```
help find
```

begins displaying the data on FIND.

EUREKA often pauses at various times during the display and prompts with a "*". The user may then hit <return> to continue, type "k" to stop the printing, or give the name of another topic. The current one is then stopped, and EUREKA moves on to the new topic. This is handy when the information in one triggers some thought covered in another.

In addition, help is available at many other times, such as when a prompt is given in browse mode. The specifics depend on the particular command.

2.12 LP command

This command sets a flag telling EUREKA to direct future output to a printer for hard copy instead of to the terminal. Using it again returns the system to normal output, making the command a "toggle switch". The command, simply "lp", affects PRINT, TERM and FREQ.

LP should not be used by the typical person, as it can generate lots of output. The printer may not be by the terminal, either, making the copy unavailable.

2.13 Make command

Query sets are usually formed by find commands. Explicit construction of new sets out of existing sets and documents is accomplished using MAKE. These new sets are given query numbers and are equivalent to ones built by FIND. The full make command syntax is

```
make <set expression> = <set name> " comments "
```

Only "make <expr>" is required, with the rest optional, as it is with FIND. The expression rules match those discussed in Section 2.2.2. Simple examples are

```
Query #10
#make 3 or 4
...
Query #11
#make 2 and 5 minus [2,10]
...
Query #12
#make last-10 = pets
```

The first constructs query set #10, combining documents which are in eithe set #3 or #4. In the second case (notice the higher query number), only those documents which are in both query sets #2 and #5 are included. Documents 2 and 10 are not included, even if they are in both sets #2 and #5. The third makes query set #12, assigns the name "pets" and includes those documents in set "last" (that is, set #11), except for those also in #10. The net expression is the same as "(2 and 5)-[2,10]-(3 or 4)", but parentheses are invalid here. Otherwise, the operator precedence of search expressions applies, with MINUS th same level as OR. Thus A+B-C*D means ((A+B) - (C*D)) and not (((A+B)-C)*D) or some other combination.

MAKE can build complex sets, if terms and operators are arranged properly. Sometimes splitting into two or more commands is still needed. To get the equivalent meaning of query #12, one request of

```
make 2 * 5 - 3 - 4 - [2,10]
```

would do the job.

Query sets formed by MAKE are not exactly the same as FIND results because MAKE produces no search terms. Therefore, a "PRINT <context>" request using a MAKE set will not output anything since PRINT would have no terms to search for and display! MAKE, then, is useful for making sets to feed to FIND through the "from" clause.

At the time of this writing, this weakness is being removed. A search expression will be constructed out of the expressions from the sets used in the make request.

2.14 Message command

A user with a comment or question about EUREKA may express it with the message command. The EUREKA system people will read the messages and leave replies in the help file "reply" (i.e. access using "help reply"). MESSAGE has both a long and short form.

The longer one is used by simply typing "message". EUREKA then gives brief instructions and prompts with "*". The user types the first line of his message and is then prompted again. He may continue to enter more lines, if desired. Hitting <return> right after the prompt gives an empty line and causes MESSAGE to exit.

The shorter form skips the instruction phase and allows the user to type his first line on the same line as "message". Examples of each type are

```
Query #9
#message
*this is a message
*with no meaning
*<return>

Query #9
#message can I change my user
*name to something else?
*<return>
```

2.15 T command - see thesaurus, Section 2.18

2.16 Users command

Users is a command good for curiosity about who is on EUREKA. By typing "users", the user sees a list of all the people currently signed on to the system, along with each person's terminal number and the time of his last command.

2.17 Words command

Evaluating a set of documents is time-consuming and difficult. The words command exists to provide aid in this area. WORDS extracts and displays keyword from a document set. The words should carry considerable information about what the documents really discuss. By viewing these words, we hope the user can evaluate her set more easily, or at least get some feedback on possible terms t incorporate in her next query attempt.

A command looks like

```
words <set name or number>
words <document list in brackets>
words <document range in brackets>.
```

If no argument appears, the "last" set is assumed. EUREKA produces a list of documents from the parameter and submits the list to the word extractor. Some sample commands are

```
words irs            (look up docs in set "irs")
words 4              (look up docs in query #4)
words [4,8,17,33,48] (use these 5 specific docs)
words [10..20]       (the 11 docs from 10 to 20)
words                (same as "words last").
```

WORDS does much more than just display words, but let's start there. The words are divided into two classes:

1) Those with high document counts (i.e. appear in many documents within the set).

2) Those with high <u>average</u> frequencies. The average is the total frequency of the word within the set divided by the number of the set's documents that contain the word.

Suppose WORDS must analyze some set with 10 documents and that tax appears 6,3,5,2,6 and 8 times in six of the documents and never in the other four. Its document count is six and its average frequency is (6+3+5+2+6+8 = 30/6, not 30/10) five. Studies show that the words with high document counts and/or high average frequencies tend to be the content-bearing words in a set, provided that "noise" like prepositions is deleted. In fact, WORDS assumes that this modification of the input source was done previously. The user will probably notice that few words displayed ever have suffixes - most are in singular, root form.

While checking for "noise", the database constructor also tries to collapse existing words to root form. The idea is that multiple forms of the word still represent the same word and that a more accurate frequency picture of the root word's use will be obtained by the combination. For example, "manage", "manages" and "managing" would be combined as three occurrences of "manage". This allows a word with several forms to add the respective frequencies together and make the keyword list when the individual forms might not have.

EUREKA shows the words sorted by count, not alphabetically, and up to 25 at a time. Therefore, the 25 most common words appear, as well as the 25 words with the highest average frequencies. (A word may be in both lists.) A sample request with a query set of 32 documents is

```
Query #10
#words 8          (query set #8)
documents: 32     (set has 32 documents)
*** words that appeared in many documents:
SYSTEM-22       HOSPITAL-17     MEDICAL-13      PROGRAM-13      DATA-12
PATIENT-11      AUTOMATE-10     BASE-10         LANGUAGE-10     PROGRAMMING-10
ADMINISTRATE-9  CLINIC-9        DESIGN-9        PROCESS-9       TEXT-9
MANAGE-8        DEVELOPMENT-7   HEALTH-7        SONS-7          COMPONENT-6
DOCUMENT-6      PHYSICIAN-6     STYLE-6         TECHNOLOGY-6    APPROXIMATE-5

*** words with high average frequencies:
ANTIBIOTIC-8    PROGRAMMING-7   EPISODE-5       MEDICAL-5       MYCIN-5
PROGRAM-5       SEGMENT-5       STYLE-5         ALGORITHM-4     CLINIC-4
HOSPITAL-4      NETWORK-4       ORGANISM-4      SYSTEM-4        BRAIN-3
NEIGHBORHOOD-3  PATIENT-3       PROVERBS-3      SORT-3
SPECIFICATION-3                 STRUNK-3        ACQUIRE-2       AMHT-2
APPROXIMATE-2   AUTO-2
Please hit <return> or enter code when ready *   (return was pressed)
Choose one of:
        0 = exit (<return> also means exit)
        1 = exit and save doc list
        2 = change the document list
        3 = see more words (if possible)
        4 = see original word list again
        5 = get help
*2                    (to be continued later)
```

This initial output shows that "system" was the most common word, appearin in 22 of the 32 documents. Note it also made the frequency list with an average of four occurrences per document for the 22 it was in. The highest average goes to "antibiotic", which could not have been in more than four documents or it would have made the other list, too. A user presented with this output might notice that the set contains two distinct subsets - medicine and computer programming. We will use this fact shortly, after explaining what the choices mean.

Choice 0 - leave WORDS and return for another query.

Choice 1 - save document list. If a user entered a document list by hand produced a narrow set using option 2, he may want to save the documents as a query set. This allows studying the set with FIND and PRINT. WORDS builds a query set equivalent to one generated by a search, only the search expression will be empty.

Choice 2 - change list. This option can be extremely useful in narrowing the list closer to the target answer. By selecting a subset of the documents that contain particular words, the user removes unwanted documents. (Choice 1 then needed to save the freshly made list.) The selection functions as either AND or OR, depending on a later decision. For AND, a document survives only if it has <u>all</u> the words chosen, while OR means take the document if it has any of the words. An example of this option is below.

Choice 3 - see more words. The original display includes only the top wor in each category. WORDS does have more than that available, so this option fetches words from farther down each list. The categories and interpretation remain unchanged.

Choice 4 - reset. After using option 3, a user may want to see the "best" words again without re-doing everything. This choice just backs the display up to the beginning.

Choice 5 - help. This prints a brief message. To get more complete help, the user should exit WORDS and give a "help words" command.

Continuing with the example, suppose the user wants to select the documents discussing hospitals or medicine, throwing out the computer articles. To get that, also suppose he decides to accept one of the 32 documents if it has "hospital" or "medicine" or "antibiotic" in it. Remember that only the given set gets involved - the rest of the database is ignored.

To start, he already entered option 2 above. The following output would be generated:

```
Now select the documents to include in the new list
but first, type 0 to "and" the words
       or type 1 to "or" the words
       or type 2 to back up to previous choices   (i.e. change mind)
*1    (we want to OR the three words)
The words displayed above will be reshown. For each word,
type y to include it, <return> to discard it,
or k to exit the section
SYSTEM *           (<return> was pressed)
HOSPITAL *y
MEDICAL *y
PROGRAM *k     (we got the 2 words from this part. K here means
                skip to the high average frequency section)
ANTIBIOTIC *y
PROGRAMMING *k     (we don't need any more words from this section.
                    K now means exit word selection and resubmit
                    the new document list for processing.)

documents: 19     (19 of the 32 had at least one of the 3 words)
*** words that appeared in many documents:
SYSTEM-19      HOSPITAL-17    MEDICAL-13     PATIENT-11     DATA-10

        (and so on ...)
```

After the second display is finished, the user gets to choose one of the six options again. The identity of the original 32 documents is gone as far as WORDS is concerned. It only knows the 19 it received on the second pass.

If a query set is submitted to WORDS and a user decides to narrow the set, the actual set is not deleted. WORDS merely "forgets" what the original documents are.

2.18 Thesaurus feature

Many sections preceding this one referenced the "thesaurus". This complex feature will now be discussed. The user should remember that this entire section may be ignored if desired, but the thesaurus can be extremely helpful in making more efficient use of EUREKA. We do recommend not employing the thesaurus until the user is comfortable with most of the rest of EUREKA.

English is a diverse language with many words very similar in meaning to other words. Knowing these synonyms greatly increases the effectiveness of search expressions. By defining a class of synonyms, the user can request that by putting one member in a find command, the entire synonym will be fetched and each word within it will also be searched for.

This optional feature is controlled completely by the user. Each one gets his own personal thesaurus to build, modify and use as he wishes. The commands available are ENTER, CHANGE, DELETE, PRINT and T. Before discussing the specifics, we will define a synonym and give examples.

1) A synonym class has up to three sections: names, terms and an expression.

2) The sections must appear in one of these combinations:

   - name(s) and term(s)
   - name(s) and expression
   - name(s) and term(s) and expression
   - term(s)
   - term(s) and expression

3) A term is a EUREKA search term (see glossary) which may be used

to reference the synonym. This is correct for names, too, but when a synonym is expanded in a search expression, terms are included and names are not. The terms are ORed together, while names serve only as labels. These terms/names may include quoted phrases, prefixing and suffixing as usual. Using these additions is recommended, except for prefixing. When looking for matches between terms and thesaurus words, quotes and universal characters (i.e. "?") are ignored.

4) An <u>expression</u> here is included in any synonym expansion. It does not get expanded itself, and neither do the terms mentioned above. The expression may be a full expression as in find commands, complete with "in <context>" parts. Because of this power, and the interpretation of "in" by the system, the expression is surrounded by parentheses. The parens "cut off" the influence of the "in", keeping it confined to the thesaurus expression part. The words within the expression are not stored individually in the thesaurus, so a class that has an expression must have at least a name or term so that it can be referenced.

<u>Examples</u>

Now for some examples, assuming the user is doing FIND commands with the thesaurus turned on.

<u>Synonym 1:</u>

names: x

terms: y, z

expr: a + b and c

The commands "f x", "f y" and "f z" would each result in an effective command of "f y or z or (a or b and c)".

Synonym 2:

names: x

expr: a or b

"Find x" for synonym 2 would turn into "find (a or b)".

Synonym 3:

names: x,y

terms: z,w

"Find x", "find y", "find z", or "find w" would produce the same result, a query of "find z or w".

Synonym 4:

terms: x

expr: y and z in sentence

"Find x" would produce an expansion of "find x+(y and z in sentence)". Note that the "in sentence" does not apply to the "x". More restrictions may appear as in "find x in title", which would expand to "find x+(y and z in sentence) in title. The title context affects both "x" and the "y/z" pair.

Synonym 5:

names: a

expr: a in title + (b*c in s)

If a user wants all terms in the class to be in some restrictive context, then they must be moved to the expression field. As #4 showed, the term list is ORed with the expression, but the list carries no context information for the terms. This last case shows the full power of an expression by having everything contained in it, with one term preserved in the name section for reference purposes. "Find a" would become "find (a in title + (b*c in s))".

EUREKA places some restrictions on the number of classes and terms permitted, and on the size of a class. Only the latter should matter, as an expanded synonym class (complete with operators) cannot exceed 256 characters. If, while defining or updating a class, some restriction is reached, EUREKA tries to prevent changes. EUREKA should catch any major problems such as no more space in the thesaurus, but if such a warning appears, the user is advised to check other synonym classes, and he probably should delete some data.

2.18.1 Enter

ENTER's purpose is constructing new synonyms. After typing "enter", the user is prompted for the three parts of the class - names, terms and the expression. A response of <return>, and thus a null line, means no definition for that part is desired. For names and terms, the user enters a list of words separated by spaces and/or commas. After all the parts are typed in, the legality of the new definition is checked.

Next, each of the names and terms is examined for existing use. Should at least one exist already, the user is asked whether he still wants the new definition entered. If so, or if no conflicts existed, the new class is added to the thesaurus. On a conflict the previous entries are not displayed, but the user may easily cancel the request, check them using PRINT SYN and then decide later what to do. Some examples of ENTER and the other commands will be given later.

2.18.2 Change

Once a particular class has been defined and the user has evaluated it, he may want to modify parts of the class. The change command exists for this purpose. Its format is: "change syn <synonym name>", such as "change syn x".

If the name/term given exists in the thesaurus, its definition is displayed and the user is prompted for action. If the term has multiple meanings, then the various ones are displayed until the user picks the one he wants to modify. When prompted for a response following the display, the user must choose one of the subcommands listed below. If the subcommand is legal, the necessary action is performed and the modified synonym is displayed. Prompting is done again, and this cycle continues until either the user exits with a null command or commits a major error. The subcommands are:

- <return> : exit the change command
- help : display information on the change command
- delete <name/term> : remove the given word from the class
- delete expr : remove the expression from the class
- add name <name> : add another name to the class
- add term <term> : add another term to the class
- add expr <expression> : add or replace the expression
- change <name/term> to <name/term> : a substitution of one by another
- change expr to <expression> : same as add expr

These commands preserve the legality of the class. For example, if the synonym has one name and an expression, then deletion of either part would violate class rules. Thus the deletion would not be performed.

2.18.3 Print

Many thesaurus requests display definitions of synonyms to the user. The print command allows explicit requests to see particular classes. Its format is simply "print syn <name/term>". If the name or term exists, all of its definitions are shown, with prompting after every two. This prompting makes easier viewing for those few classes that may have many definitions, and for the second case of the command, "print syn all". The "all" keyword dumps the entire thesaurus on the screen, in the same format and pausing after every pair.

2.18.4 T command

The t (as in thesaurus) command is only for setting states, such as turning thesaurus use on and off. "T on" and "T off" do just that, and when the thesaurus is off, search expression terms are not looked up. When the thesaurus is on, lookup and expansion is automatic and requires user action only when a term has multiple meanings. He must then choose the desired class.

Even when the thesaurus is on, it can temporarily be overruled within the search expression. Suppose a user has an expression of two words, one of which he wants checked in the thesaurus, and one he doesn't. Turning it on assumes the user wants both, but by typing "@" as the first character of the term not to look up, the thesaurus will not be consulted. Thus "f tax and @income", entered with the thesaurus on, would look up "tax" and not "income". Note that with the thesaurus off, the "@" is treated as a normal character.

T has one other form, also related to find commands. If the user wants to see expansion of terms within search expressions, "t display on" is needed. The condition stays on until either a "t display off" command is entered or until he exits the system. When DISPLAY is not on, nothing is printed beyond a normal FIND, so the thesaurus use is transparent, unless the user has to resolve a conflict. Of course, the thesaurus must also be on for DISPLAY to work.

2.18.5 Delete

Our final command is DELETE, which removes entire classes from the thesaurus. Its format is: "delete syn <list of names/terms>", or "delete syn <name/term>". That is, deletion of several synonyns at once is possible in the first case. The definition of the class is displayed and the user is asked if he still wants it deleted.

This display helps to avoid mistakes, allows for changes of heart and resolves conflicts among those terms with many definitions. A multiple definition case may end up with no changes, all the definitions deleted or with some removed and some kept.

When a class is deleted, each name/term in it is also looked up and reference to the now non-existent synonym is removed. If this class is the only one it is in, then that term is likewise purged.

By using "delete syn all", the entire thesaurus is cleared - all names, terms, classes and space are freed.

2.18.6 Sample EUREKA user session

System responses are underlined.

```
Query #1
#enter                                  (example 1)
names:
*<return>
terms:
*tax taxes taxpayer? income irs 'internal revenue service'
expr:
*return and file


Query #1
#print syn irs                          (ex. 2)

term(s):
TAX TAXES TAXPAYER INCOME IRS 'INTERNAL REVENUE SERVICE'
expression:
(RETURN AND FILE)


Query #1
#enter                                  (ex. 3)
names:
*income
terms:
*wages salary pay employ?
expr:
*<return>

INCOME has a definition already
```

```
Should the new synonym be added?
*y


Query #1
#t on             (turn thesaurus on, ex. 4)


Query #1
#t display on      (doing a FIND shows expansions, ex. 5)


Query #1
#f salary                                  (ex. 6)

SALARY expands to:
WAGES+SALARY+PAY+EMPLOY?
... result of find command ...


Query #2
#f income     (income has 2 definitions, ex. 7)
INCOME has a multiple definition. Please pick one
... the definition with tax (example 1) is shown

Is this the one?(y or n)
*y

INCOME expands to:
TAX+TAXES+TAXPAYER?+INCOME+IRS+'INTERNAL REVENUE SERVICE'+(RETURN AND FILE)
... result of find command ...


Query #3
#change syn tax                                (ex. 8)
changes will be made to:
... the class definition with tax (ex. 1,2) is shown


Please enter request-help available
*change expr to return and file? and fail? in paragraph
synonym now:

terms:
TAX TAXES TAXPAYER? INCOME IRS 'INTERNAL REVENUE SERVICE'
expression:
(RETURN AND FILE? AND FAIL? IN PARAGRAPH)

Please enter request-help available
*delete income
synonym now:

terms:
TAX TAXES TAXPAYER? IRS 'INTERNAL REVENUE SERVICE'
```

```
expression:
(RETURN AND FILE? AND FAIL? IN PARAGRAPH)

Please enter request-help available
*<return>         (exit change sequence)


Query #3
#delete syn income                (ex. 9)
(Income did have a multiple definition (ex. 1,3), but
income was deleted from one class (ex. 8), leaving only
one definition.)

name(s):
INCOME
term(s):
WAGES SALARY PAY EMPLOY?

Do you really want it deleted?
*y


Query #3
#

... and so on for the rest of the session ...
```

3. Comments on EUREKA

3.0 General search ideas

In broad terms, many things influence the effectiveness of EUREKA. These are

1) The user's understanding of what the system can do.
2) How well the user can define his problem to himself.
3) How well he can express the problem to the system.
4) How good the database is in the topic of interest. (thorough coverage, up-to-date, etc.)
5) How big the database is, both the number of documents and the size of the documents.

The first three cover the man-machine interface, affecting input to EUREKA. The other two consider the basic capability of EUREKA to answer the question completely and efficiently.

3.1 Search strategy

Because EUREKA is interactive, it gives quick response to requests. With such interaction available, the user need not try to formulate a detailed query that gives perfect results in one try. One should aim for an iterative strategy, gradually narrowing down the set of documents to get an answer.

When beginning investigation into a subject, the first query should aim for high recall, which means trying to retrieve as many of the potentially relevant documents as possible. This first set should give an idea of what the user faces. If the set is large, the query should be modified for a more restricted retrieval. If the set is not too big, the user should browse through some or all of the documents and evaluate the contents.

Usually the set is too broad. In this case, any evaluation (e.g. reading the text of a few of the documents) should have provided feedback about what changes in the search expression are needed. Since the target information is probably contained in that query set, the "from last" clause is recommended. By using "from" in a new query, the document set should be narrowed closer to the answer. By repeating the process as often as needed, the user should eventually discover the results hoped for.

Sometimes a few searches can be carried out at once. Say the user wants data about some general subject "X", but not when it also discusses "Y". MAKE is useful here. The first search series is performed and then the second, independent of the first. A make command, such as "make x-y" does the selective deletion of subject "Y" documents. Generalizing to more complex make statements provides even more power.

3.2 Suffixing

Using a "?" at the end of words is usually very handy, especially in early queries where high recall is preferred. The term command can show what words will be searched for through the use of suffixing. Sometimes a few undesirable words are included, as with "taxicab" in the Section 2.9 example.

If so, conduct a search for those specific "bad" terms within the final result set, creating a new set. By then using a MAKE and deleting those new documents, their influence is removed. Of course, if one of the "good" words also appear in those documents, its occurrences will be lost. Therefore, this type of strategy requires caution.

3.3 Search failures

Even if the database has the needed information, that data can still elude diligent searchers. These failures come from several contributing areas:

1) Forgetting proper search terms. Since EUREKA indexes every word, the user may have to think of every word needed to cover a topic. Failure to include a major word in the search expression may remove the only chance to access relevant documents. As a trivial example, a search for articles on pigs could

fail if a user forgets that an article might reference "hogs" instead of specifically "pigs". A related problem is case 2.

2) No specific term available. In systems with a small controlled vocabulary, the indexer can think up a term suitable for representing the document. Continuing with the above example, suppose several hog diseases are discussed by name, without ever using a general term like "disease". In a controlled system, an index term like "hog diseases" might be used, but in EUREKA, where only exact words from the data are indexed, the user would have to think of particular diseases to list or try other words that mean disease. More likely, though, is that the document won't be retrieved.

3) Incorrect search expression. Even when the right terms are found, some searches fail because of improper logical use of AND, OR, IN, FROM and parentheses.

4) Poor analysis of the problem. This is more common when someone conducts a search for another person. Key ideas can easily be lost in the discussion. In a self-search, the user should carefully analyze exactly what he wants and define the target logical areas. For example, a search for "pollution caused by disposal of nuclear power plant wastes" would first be analyzed as having three distinct subtopics - nuclear power plants, waste disposal, and pollution. This may seem like a trivial step, but the same topic expressed as "land and water contamination by radioactive wastes" may not be split up the same way, even though the subject is effectively the same (assuming the user implied nuclear power with radioactivity). By the time the topic is translated into a search expression, an error in this conceptual phase can seriously weaken the search.

5) Recall versus precision (see glossary). A search usually involves a tradeoff between these two measures. Before formulating the search request, the user should choose a strategy for either high recall or high precision. Using

many general search terms with suffixing, but without restriction of contexts and ANDs tends to increase recall at the expense of precision. If a small number of very relevant documents is satisfactory, then a strategy of very specific terms, quoted phrases, contexts, and so on, may be appropriate. Potentially relevant documents will probably be lost. With the iterative searching of EUREKA, trying for a high recall first and then narrowing down to improve precision usually works best.

6) Suffixing. Failure to include suffixing hurts recall when a term appears in a document only in some suffixed form and not in the root form given. However, trying for all endings often hurts precision by dragging in improper words with the same root.

7) Homographs. This hopeless problem comes from words with the same spelling but different meanings. Fortunately, it is not very serious, but an example is "bow". In a broad database, it could refer to ships or to arrows. Including another relevant term in the expression can provide the necessary restriction, such as "bow and arrow". That phrase is unlikely in an article on boats.

8) Failure to include peripherally related words. This idea is more esoteric than others. Consider including "detroit" as a term for a search on an auto industry topic, or "mental health" for a search on "mental illness" since illness is a condition of health.

9) False coordination. In a search for "tax", any document with "tax" is retrieved, even one with a single occurrence in an incidental passage. The problem, then, is words that aren't really relevant to the topic of the article, but are found anyway.

10) Incorrect term relationships. This problem is a serious one, since the appearance of two terms in a document does not mean the occurrences are related.

An expression like "a and b" assumes the dual appearance implies a relationship. Suppose "tax and rate and auto?" was an expression. A document about changing import tax rates on automobiles probably would be found, as would an article covering tax money wasted on government cars with poor mileage rates. If the first case is what the user had in mind, then the second would not be relevant. This problem of terms not appearing the way expected is the major force behind the use of contexts and quoted phrases. The closer relationship implied by them removes many of these improper matches. The remainder just cannot be handled without a more sophisticated system.

4. Glossary of terms

Context - each document is divided into several parts, such as author, title, references and so on. Such a division is called a context. All documents within a particular database have the same contexts defined, although some parts might be empty. For example, a document might have no references. The exact name of each context depends on the database. The contexts "sentence" and "paragraph" may appear arbitrarily many times within a document, but all others may appear only once.

Document - a database is separated into a series of documents, each assigned a number. What constitutes a document, such as one journal article or one chapter of a book, varies between databases. Every word within a document is added to the index and can be searched for.

Index - a search for a term consults the index to see if that word exists. The index contains every word that appears anywhere in the database, plus information about how often and in which documents.

Precision - a measure of search effectiveness. Mathematically defined as the number of relevant documents retrieved divided by the total number retrieved. Intuitively, it is the proportion of material found that is actually useful.

Query - a command to EUREKA to execute some request. Each query has a number, but only those queries that generate a set of documents save the result and change the query number.

Query name - a label can be attached to the query results. It is then equivalent to the number. The name must start with a letter and may have up to nine more characters, each being a letter or digit.

Query set - the list of documents retrieved by a FIND or built with MAKE. A query set print includes the search expression, query name, the documents in the set and comments. Each set has a number to distinguish it from others.

Recall - a measure of search effectiveness. It is defined as the number of relevant documents retrieved divided by the total number of relevant documents in the database, ignoring how much extra junk was also retrieved. It measures the proportion of the good documents actually found. Getting high recall often forces low precision.

Search expression - FIND examines its search expression to determine what terms to search for and in what combination. The expression is a series of terms separated by the operators AND and OR. AND has higher priority than OR.

Search set - when a search is conducted, the set of documents actually examined is called the search set. The "from" clause allows a combination of query sets and documents to be specified as the search set. This confines searching to the documents of the search set and not the entire database.

Synonym class - an entry in the thesaurus, composed of terms and an expression. A class should represent a set of words similar in meaning, or can be considered shorthand for some commonly used search expression.

Term - an operand in a search expression, also what EUREKA can explicitly search for. A term follows a strict set of rules:

1) a maximum of 32 characters.
2) if the term contains more than one word, i.e. has spaces within it, it should be surrounded by single quotes. To include quotes within a quoted term, two consecutive quotes are needed. For example, "Murphy's Law" would be entered as 'murphy''s law'.
3) if the term is unquoted, it is terminated by the appearance of a blank, "+", "*", "&", ")" or the end of the line. Note that single

quotes can be embedded in unquoted terms. The terminators like "+" may appear inside quoted terms as part of the term.

4) a "?" at the beginning of a term signals prefixing, and one at the end means suffixing.

5) when the thesaurus is on, a "@" at the beginning of a term means don't look up this term in the thesaurus.

6) If a term is quoted or if it contains characters that are not just letters and digits (e.g. 27,610 has a comma), EUREKA extracts the substrings containing only alphanumeric characters. A document containing all such substrings is then searched for an exact match with the complete search term. (e.g. 27,610 is indexed as 27 once and 610 once).

PART 2 - EURUP

1. Introduction

EURUP is a system for constructing, updating and maintaining databases for EUREKA. It is operational and useful, although incomplete, and still undergoing development. This part of the report documents the structure of EUREKA databases and the working portions of EURUP, as well as a few commands that are being finished or modified. It is meant to be a guide to the use of EURUP for constructing and maintaining databases.

Some of the dynamic properties of databases used by information retrieval systems such as EUREKA include steady or erratic growth over an extended period of time, occasional or periodic removal of obsolete data, and correction of random errors in the data (e.g. typographical). Hence there is a need to be able to modify or extend a database. But updates cause the database to become disorganized, which results in EUREKA taking longer to access desired data because it is not in the best location. Therefore, there must also be the ability to restructure the database, perhaps employing new hardware, as the database grows or improved techniques are applied.

The primary reasons for implementing EURUP are to provide easy procedures for constructing a database from raw text and to allow easy database updating. Listed below are other considerations in the design of the database structures and EURUP.

1) Unique, well-defined declarations and data accessing procedures will be provided.
2) The integrity of database structures will be maintained at all times.

3) The possibility of on-line updates should not be ruled out. This is an interesting problem and ideally, a large information retrieval system should be available 24 hours a day. Maintaining data integrity is a first step in this direction.

4) Reorganization of the various database components, the index in particular, is provided so that search times can be minimized.

5) Databases are capable of being split and stored on more than one volume (i.e. disk pack) in order to handle databases larger than a single volume.

6) Some of the data files in a database will have variable block sizes.

7) The updating system should make future data structure, format and procedure changes easier and less disruptive to the user environment.

The next section details the structure of a database, discussing the various system files needed and the database components stored in these files. Some knowledge of the DOS filing system is assumed and will undoubtedly be necessary in order to use EURUP and this manual. Section 3 talks about the inputs and outputs that EURUP uses and generates. In particular, the format of text to be input to a database is completely defined. Section 4 describes the functions of EURUP and gives the syntax and operation of the command language. The final section gives advice and some examples of using EURUP to build or modify a database.

2. Database Structures

A EUREKA database is a collection of "documents". Each document contains nothing but text and is broken up into a fixed number of sections, such as title, author, abstract, body, references, etc. As will be seen, the number of sections and their names are database parameters, selected to suit the data and/or the database manager. The documents are assigned unique numbers for addressing purposes. Associated with each document is an optional "vocabulary file" which contains a list of words and the number of times they appear in the document.

The "database index" is a complete list of every word that appears in any document of the database. Each "term" in the database index has an associated "postings file" of document numbers, indicating which documents contain that term. With each document number in a postings file are the term frequency and context flags. These indicate which sections of the document the word appears in and how many times altogether. In satisfying a "find" request, EUREKA first consults the database index, to determine which documents contain the desired logical combination of search terms. In some cases, the database index will not yield a final answer to the search request, but will give a list of documents that may satisfy the request, guaranteeing that no other documents in the database do. EUREKA scans the complete text of those documents to determine which ones actually satisfy the search request. For example, to find "income tax", a document with both "income" and "tax" must be checked to see if the terms are adjacent.

Databases are stored in several interrelated files kept in user code [1,5]. The "database directory" contains symbolic pointers to the other "extents" of the database. Each extent is a contiguous file, and there are three types: index, text, and vocabulary. Other components of a database are stored in files that have the same name as the database directory and an identifying file name extension.

2.0 Database Directory

The database directory is a relatively small contiguous file which is used to locate the other extents of the database and also contains the document and vocabulary directories. The file name extension for the database directory is "DB". The database directory always takes an odd number of physical blocks (say 2n+1).

The first block of the file (block 0) contains the "extent directory", which is a list of symbolic pointers to the other extents of the database. Each pointer indicates the volume (i.e. device and unit) where the extent is stored and the name of the file. These pointers are set by the create command when the database is created, and can be modified with the change command.

Blocks 1 thru n contain the "document directory". The jth entry in the document directory points to the start of document j, beginning with j = 0. Similarly, blocks n+1 thru 2n contain the "vocabulary directory", and the kth entry points to the start of the vocabulary file for document k. Note that the vocabulary directory exists even if the database contains none of the optional vocabulary files.

At database creation, a value for the maximum number of documents in the database is input. It determines the sizes of both the document and vocabulary directories, and hence the size of the database directory file.

2.1 Database Index

Every database has exactly one database index extent, referenced through the database directory. This extent contains the "terms file" and the "postings files". The terms file is indexed using ISAM techniques for quick word searches.

Block 0 of the extent is a header of control information and statistics for the extent. The extent also contains a bitmap so that space for terms and postings can be dynamically allocated.

The terms in the index are restricted to being alphanumeric strings of any length less than 32 characters. Special characters are not indexed. Each term is assigned a unique 24-bit integer value in lexicographic order. Actually, the first character of a term forms the high 8 bits of the term number. The low 16 bits are selected with more or less even spacing over all the terms that begin with a particular letter or digit. This leaves gaps which usually allows new terms to be inserted without term number conflicts. The use of term numbers and some of the associated problems are discussed in subsequent sections.

Normally, each term entry contains a pointer to the start of its associated postings file. Each posting entry represents a single document and holds a term frequency for how many times the term appears in that document. The value is limited to a 7-bit field, and so it is set to 127 if the term appears more than 126 times. Truncation causes inaccuracy only for very frequent terms, which typically bear no information anyway.

In normal English there are always a few words that appear many times and in many documents, like "the". These words have little information content but have large postings files. If a postings file gets to be a certain size it is deleted, but the term entry remains with a special nil postings file pointer. This is taken to mean that the term appears in every document (even though it may not). The deletion threshold is currently fixed at 500 postings, but in the future should be made a parameter of the database.

2.2 Text

The text of each document is stored in a "document file". Each document file contains a "document context index" and the text of the document. The document context index is stored in the first text block of the document and contains a pointer to the beginning of each section of the document as well as a pointer to the end of it. The text blocks are doubly linked so that EUREKA browse mode can scroll forward or backward through the text.

Document files are stored in text extents, which are large contiguous files. A database can have a variable number of text extents, but must have at least one. The size of each text extent is set when it is allocated. All text extents have the same structure. The first block contains control and statistical information. A bitmap is stored at the end of the extent. This is used for dynamically allocating text blocks from the remaining space. If all the text extents fill up, either they may be enlarged and reorganized, or an additional text extent may be allocated. The extent directory (part of the database directory) points to each of the text extents in the database.

The document directory (also in the database directory) is used to look up arbitrary document files. Each entry in the document directory specifies in which text extent the document file is stored, as well as the relative address of its first block of text within the extent. Note that a document is completely contained in one text extent. It cannot be split across extent boundaries.

2.3 Vocabularies

Each document's vocabulary file is a sorted list of term numbers and frequency values for selected words from the document. A vocabulary file may be created during document insertion, or it may be (re)generated for an existing document at any time.

Vocabulary files are stored in vocabulary extents, which like the text extents are large contiguous files. A database can have a variable number of vocabulary extents, including none at all, in which case no vocabulary files may be created. The size of each vocabulary extent is set when it is allocated. They have the same structure as the text extents, but contain data blocks for the vocabulary files rather than text blocks. Vocabulary extents can be enlarged, reorganized and allocated just like the text extents. The extent directory (in the database directory) points to each vocabulary extent in the database.

The vocabulary directory (also in the database directory) is used to look up arbitrary vocabulary files. Each entry in the vocabulary directory specifies which vocabulary extent the vocabulary file resides in, as well as the relative address of its first data block. Like document text, a vocabulary file cannot be split across extent boundaries.

Vocabulary files contain term numbers instead of the actual term text because of the storage savings and the quick processing made possible for some EUREKA programs. After processing vocabularies, the term numbers are used to search the terms file in order to display the actual words. Originally a vocabulary file was to represent a document exactly, having all and only those words that are in the text. However, the needs of EUREKA commands dictated otherwise. During vocabulary construction, the document text now passes through a "stemmer", which converts most suffixed terms into root forms. The frequency count for the root is a sum of the counts for any suffixed forms in the document. For example, suppose a document contains "manage", "managing" and "managed". Its vocabulary file will contain only "manage", but the frequency field will include the occurrences of all three.

A vocabulary entry is always a "real word", meaning a word in the database index. A root produced by the stemmer is rejected if the stem cannot be found in the index. Instead, the original word form is entered in the vocabulary. One major reason for this decision is that each vocabulary entry needs a term number, but only indexed words are assigned numbers. Another reason is that EUREKA users can read vocabulary contents indirectly through the words command, so the entries should be real.

Further vocabulary modification comes through use of the "stop" and "save" lists. The vocabulary builder throws out all terms under four characters and those that begin with a digit or are on the stop list. The save list prevents some words from being deleted and prevents unwise stemming in other cases. See Section 2.5 for more details.

Stemming, stopping and saving can be avoided if a vocabulary in the original scheme is desired. The controls exist as program source code switches, though, making necessary a new assembly, etc.

2.4 Context Definitions

The context names for a database are defined in a special file. This file has the name of the database and an extension of "CTX". It is a simple text file, created and modified with the editor. As mentioned before, a document can consist of up to 12 sections. Any subset of sections can be given a context name and any section can be included in any number of context definitions. The definitions are used by EUREKA for two purposes. One is to translate a user's context specifier into a representation for the actual sections of the documents which are to be searched or printed. For example, "cite" might be defined as sections 1,2,3 and 5. A EUREKA command of "find knuth in cite" would cause EUREKA to search for all documents that contain "knuth" in any of sections 1,2,3 or 5. Anytime EUREKA expects a context name in a command, it will only accept a name that is in the context definition file. Obviously, EUREKA keywords, such as "from" or "last", cannot be used as context names.

The other purpose of the context definition file is to define labels to be printed as the various sections of a document are displayed for the user. For example, if "title" is defined as section 1, then "title" will be used as a label anytime the text of section 1 is displayed.

How the text of a document is broken into 12 sections will be described in section 3, but it should be noted that all of the documents in a database will use the same context definitions, so they should all have the same logical organization. If the organization of the documents in a database only requires a few sections, then any of the 12 can be used and the remainder left empty. The empty sections need never be referenced in any of the context definitions. However, care must be taken to ensure that no data is put into a document section for which there is no context definition, or else that data will not be searchable or printable.

There is one context name definition per line in the CTX file. Each line contains three items separated by single spaces or commas, the format being:

```
<context name> <section flags> <section number>
```

The context name must be alphabetic upper case. The section flags item is an octal word of bit flags which indicate the sections represented by the context name, interpreted as follows:

bit 15:	paragraph	bit 7:	section 6
bit 14:	sentence	bit 6:	section 7
bit 13:	ignored	bit 5:	section 8
bit 12:	section 1	bit 4:	section 9
bit 11:	section 2	bit 3:	section 10
bit 10:	section 3	bit 2:	section 11
bit 9:	section 4	bit 1:	section 12
bit 8:	section 5	bit 0:	ignored

The paragraph flag indicates that searching (printing) is to be restricted to individual subdivisions of the sections being searched (printed). The sentence flag increases restriction to subdivisions of paragraphs. For example, if the context file contains the following entries:

```
SUBSECTION 100000 0
HEADNOTES 20 9
NOTES 100020 0
```

then a context specification of "in subsection in headnotes" is equivalent to "in notes", and means that subdivisions of section 9 should be searched or printed, because bit 15 (paragraph) is on and so is bit 4 (section 9).

The section number item identifies that the name of this entry shall serve as a label when printing the section. Typically there is a context name for each individual section of a document, and this name is also used as the label for printing. However, this is not mandatory. It is valid to create an entry which is to be used only as a label by setting the section flags to 0. In this case, the context name will not be acceptable in a search or print request. Also, entries that include more than one section usually are not used as section labels, and so the section number of these entries is set to 0. In the above example, HEADNOTES will be printed as a label any time text from section 9 of a document is displayed.

The order of the entries in the context definition file is only important for the determination of abbreviations. EUREKA does a linear scan on the entries, looking for the first one with a prefix that matches the user's context specification. For example, if the file contains:

```
DATA 17600 0
DATE 1000 4
```

then only "d" is required to specify the DATA context, but "date" is required to indicate the DATE context. "dat" will match DATA before DATE is found, and so it will be interpreted as DATA. This example assumes that there are no entries preceding DATA which begin with D.

The context file is read by EUREKA when a user signs onto a database. The file need not be present, in which case EUREKA reads standard context definitions from file "STD.CTX[1,5]". The actual contents of this file are:

```
PARAGRAPH 100000 0
SENTENCE 40000 0
DOCUMENT 17776 0
DATA 17600 0
AUTHOR 10000 1
TITLE 4000 2
SOURCE 2000 3
DATE 1000 4
PAGES 400 5
MISCELLANEOUS 200 6
INDEX 100 7
TEXT 36 0
WORDS 40 8
ABSTRACT 20 9
BODY 10 10
FOOTNOTES 4 11
REFERENCES 2 12
```

2.5 Stop and Save Lists

Each database has optional "stop" and "save" files of text, with names equal to the database name and extensions of "STL" and "SVL", respectively. The two files also have encoded versions, but that discussion is deferred to the stop command in Section 4.5.8.

Section 2.3 mentioned EUREKA programs that use vocabularies. They produce more meaningful results if "noise" is purged from the vocabularies, namely those high frequency words of low meaning, like prepositions, articles and common names. The stop list contains terms to delete during vocabulary construction. The list is checked after stemming, so "noise" that comes in many suffixed forms need be entered only once.

The stop list has a counterpart for saving words. Terms under four characters long (after stemming) are automatically pitched, unless they are on the save list (e.g. "tax"). Without question, most short words are useless, but this allows the few useful ones to survive. Terms beginning with a digit are also deleted, but the save list cannot prevent that. The stemmer always consults the save list first, so that specific suffixed terms can be kept and to prevent occasional unwise decisions on the part of the stemmer. For example, if one would really like to keep "automatic" from becoming "automate", then an entry of "automatic" would do so.

Detailed discussion of actually what should be stopped or saved is beyond this manual, but the list format is not. The two files have identical formats of upper case text produced by the editor or automatically through a version of the stop command. Any number of words (starting with a letter) may be on a line, separated by blanks and/or tabs. The lists must be sorted by first letter only, although full sorting is certainly allowed. For example, all the "A"s must appear before the "B"s, but any ordering *within* the "A" group is acceptable. The first entry on a line must start in column 1, and all words on a line must start with the same letter.

3. EURUP Inputs and Outputs

EURUP is an interactive system because it reads commands from a terminal. It hardly acts like one, though, since many of the commands cause the system to execute for minutes or even hours. Commands can be entered to update a database with a batch of documents or to reorganize a large database extent, requiring considerable I/O. Besides reading commands, the terminal is used for logging error, warning and statistics messages. The EURUP command language is described in Section 4. The rest of this section discusses the various input and output data files which EURUP uses and generates.

3.0 Document Input File Format

One of EURUP's primary functions is to update a database by inserting, replacing or deleting whole documents. When inserting or replacing a document, EURUP gets the text for a document from a "document input file". This is a standard text file which contains special embedded "markup" codes.

A big problem, which cannot be entirely handled by EURUP, is the generation of these document input files. Typically, text for a database comes from some external source, such as a publisher. The complete text is written in a single stream, containing markup codes used by the database supplier. These markup codes generally define various section boundaries, character code translations and/or provide formatting information for use by a typesetting system. A special formatting program must be written to break the text into individual document input files, which can then be processed by EURUP. The document file names for a given database should all start with a prefix of one or two letters, followed by a document number. They do not have to be in user code [1,5] as do the database extents. See the insert command (Section 4.3.1) for more details. This formatter must do appropriate conversions on the supplied markup codes in the original text.

The format of a document input file has been designed to simplify the formatting programs as much as possible. The EURUP markup language contains a few codes that delimit sections of text and some that only supply format information used when displaying the text. The text of the document is format free, that is, all format information for the display of the text is contained in the markup codes. The normal ASCII format control characters, such as carriage return and line feed, are treated as blanks. Strings of blanks are quashed into a single blank.

The sections of a document need not appear in numerical order and can be interleaved. For example, a document might first contain text for section 5, followed by text for section 1, and then more text to add to section 5. In addition, there is a special markup code that causes the document input processor to go back to the previous section. Paragraph markers can be placed to subdivide a section, and sentence markers to subdivide a paragraph.

An asterisk indicates that a markup code follows. Below is a list of all the codes that are used in the EURUP markup language. For the alphabetic codes, case is ignored. Some codes accept a numeric parameter, represented by n.

Document Markup Codes

*Cn	Specifies which section the following text is to be inserted in. n can be 0 thru 12. If 0, the following text is discarded, otherwise it is put into section n. If n is greater than 12, the markup code is ignored and a warning message is logged.
*C-	Leave the current section and place the following text in the section that was previously used (i.e. being processed just before the current one).
*P	Paragraph boundary. Any adjacent blanks, empty sentences or empty paragraphs are discarded.
*S	Sentence boundary. Any adjacent blanks or empty sentences are discarded.

*#n	Formatting command which forces n blanks in the text. Any surrounding blanks in the text are discarded. n can be 0 which forces the surrounding text to be adjacent even if a blank appears in the text.
*+	Formatting command which forces a new line. Text is normally formatted for display by filling output lines. A new line can be started at any point with this command.
**	Used to force an asterisk into the document text.

The document input processor starts a new document in section 0. In other words, all text up to the first *C code is discarded. Here is an example input document to clarify the use of the section codes.

```
alpha *c7 beta*c2gamma
   *c-      delta *p
 *s  rho *s omega *p*c5 sigma *c5 epsilon *c-
omicron     *c7   zeta
```

text	location
alpha	discarded
gamma	section 2
sigma	sect 5
epsilon	sect 5 (continued)
omicron	sect 5 (continued)
beta	sect 7, paragraph 1
delta	sect 7, para 1 (continued)
rho	sect 7, para 2, sentence 1
omega	sect 7, para 2, sent 2
zeta	sect 7, para 3

3.1 Statistics and Message Output Files

Many of the EURUP functions generate statistics about their data processing phases. Such statistics are normally formatted and output to standard text files. The updating commands in EURUP generally provide for the specification of a log file which is opened for extension. While processing a command, all statistics and certain warning or other informative messages are written to the log file. If not specified, the default is to log this information at the terminal. Errors severe enough to cause abortion of the current command generate messages at the terminal as well as in the log file. Other commands will be available for formatting and printing information about what is stored in the database.

The actual information written by the various commands is given in Section 4 with the command descriptions. It is conventional to specify a log file of "database name.LOG", for developing a file with a fairly complete history of the updates made to a database. As with the document input files, the log files are not restricted to user code [1,5].

3.2 Temporary Files

There are a few instances in EURUP where the CPU's memory is not large enough to hold all the data being processed. When necessary, EURUP will generate appropiate temporary disk files. Usually the temporary files will be deleted when no longer needed, but the program code may sometimes be conditionalized to save them. In either case, temporary files will have an extension of "TMP" and will not be protected, so that PIP can be used to delete any garbage that may accrue. Sometimes the user may supply a name for a temporary file and keep it for future purposes. The use and format of temporary files will be explained further as they are encountered in the command descriptions of Section 4.

4. EURUP Functions and Command Language

EURUP runs under the same operating system as EUREKA itself and may eventually be able to run concurrently with EUREKA. It can't do so now because there is no interlocking mechanism between EUREKA and EURUP. A database which is being used by a EUREKA user should not be updated by EURUP, and vice versa. EURUP has been designed, however, with such mechanisms in mind, and such features may be included later. The terminal is used to enter commands and to log error messages and statistics. EURUP does extensive error checking, and with knowledge of the database structures and operation of EURUP, the error messages should be self-explanatory.

There are two command levels in EURUP, called command mode and update mode. EURUP starts in command mode and prompts the user with a "#". In command mode, a CSI string is read which specifies a database name, a command switch, and possibly some parameters. The commands at this level operate on database extents (i.e. system files) and usually don't require access to the whole database.

One of the command mode commands causes the specified database to be "opened". Update mode is entered and the user is prompted with an "*". Opening a database involves setting up communication between EURUP and all the extents of the database. This mode is concerned with operating on a database as an entity. At this level, commands can be entered for updating or listing information about the database.

The notation used in the command descriptions is fairly trivial. Commands and keywords are shown with the first few letters capitalized. The capitals indicate the minimal abbreviation accepted by EURUP. Any letters following this abbreviation are simply ignored. Lower case items are parameters supplied by the user. An item in square brackets is optional or only applies to certain commands. Wherever a space is shown separating words, any number of spaces can be used, and in most instances one comma may also be inserted (in which case no spaces are necessary). All other special characters behave like keywords and must appear as shown.

4.0 Command Mode

Command mode commands are expected when EURUP prompts with a "#". These commands have the general form:

```
database name/command[:parameters][,file parameters]
```

The database name is a file specification for the database directory. Some commands prompt the user interactively to obtain additional parameters. Some generate information that is sent to a log file, or to the terminal if no file is specified. The commands available and the sections where details can be found are:

4.2.1	CREATE	creates a new database.
4.2.2	EXTEND	attaches an additional extent to an existing database or enlarges the database directory.
4.2.3	DETACH	removes an existing extent from a database.
4.2.4	ZERO	re-initializes a database, deleting all data from it.
4.5.1	OPEN	opens a database and causes EURUP to enter update mode.
4.4.3	REORGANIZE	reorganizes a database extent.
4.4.1	DUMP	converts an extent to a linear form by dumping it to a sequential file.
4.4.2	LOAD	reloads an extent from a previously dumped sequential file.
4.5.3	DIRECTORY	provides a listing of information in the database directory.
4.5.4	CHANGE	provides the ability to change the extent pointers in the database directory.

4.1 Update Mode

Update mode commands are expected when EURUP prompts with a "*". All update commands operate on the currently open database. The commands available in update mode are:

4.3.1	INSERT	inserts documents or vocabularies into the database.
4.3.2	REPLACE	replaces documents or vocabularies in the database.
4.3.3	DELETE	deletes documents or vocabularies from the database.
4.5.2	EXIT	closes the database and returns to command mode.
4.5.5	SELECT	specifies which text (vocabulary) extent document (vocabulary) files are to be inserted in.
4.5.6	MOVE	allows the user to move document or vocabulary files from one extent to another.
4.5.7	NUMBERS	controls term number assignments.
4.5.8	STOP	makes a compact, coded stop list from text input, or semi-automatically builds the text file.
4.5.9	UNLOCK	recovers a locked database.

4.2 Database Creation, Extension and Deletion

Creating a database involves allocating and initializing the various files, including the index, text and (optionally) vocabulary extents, the context definition file, and the stop and save lists. The database is initially empty. Update mode is then used to stuff documents into it one by one. In its simplest form, a database consists only of the database directory, index and text extents. If vocabularies are to be generated, then a vocabulary extent is needed. Additionally, stop and save list files can be created for purposes of tailoring the vocabularies.

Once created, a database is a fixed size, and hence only so much data can be inserted. However, if it becomes full, there are ways to make it larger without having to completely reconstruct it from the document input files. If the text or vocabulary extents become full, an additional extent can be allocated and attached to the database. If the index fills up, it must be reorganized and copied to a larger file.

If enough documents are deleted from a database to empty a text or vocabulary extent, that extent can be detached from the database and deleted. It may also be possible to move document or vocabulary files from one extent to another in order to make it empty so that it can be deleted. It is always possible to use system utilities to simply delete all of the files that comprise a database. There is also a convenient command that deletes all the documents and vocabularies of a database, leaving it empty as if it were just created.

4.2.1 Create

The create command allocates a database directory file and initializes extents for a database. The form of a create command is:

```
database/CReate<index,text[,vocabulary]
```

The create command will only create databases with one text and one or no vocabulary extent. The extend command attaches additional extents. The index, text and optional vocabulary extents must first be allocated, typically with PIP. The create command is then used to allocate a database directory file and set the pointers to the given database extents. No user codes may be supplied with the file specifications, for all database extents must be in [1,5]. Also, no extension may be given with the database name for this is set to "DB". The create command requires that all extents be on units that have the same physical block size (but not necessarily the same type of device).

Assuming that the specified extent files are found, the create processor requests the maximum document number and the block sizes of the different types of data blocks in the database (i.e. types, postings, text, and vocabulary). The maximum document number specifies the size of the document and vocabulary directories, and hence the size of the database directory file. This number is limited to 4095 because the document number field in postings is 12 bits. The extend command can enlarge the document and vocabulary directories of an existing database if necessary. The data block sizes must be a multiple of the physical block size (merely to convince the user not to waste space). Note that if no vocabulary extent is given, then the vocabulary block size will not be requested.

When a database is created or zeroed (Section 4.2.4), the term number assignment option is turned off. Vocabulary files cannot be inserted until term numbers are assigned. For details, see Section 4.5.7.

4.2.2 Extend

The extend command allows the user to attach additional text or vocabulary extents to a database if the existing extents become full. It also provides the capability to enlarge the database directory file. The extend command has three forms:

```
database/EXtend:TEXT<extent file
database/EXtend:VOCabulary<extent file
database/EXtend:DIRectory
```

If the "text" or "vocabulary" keyword is supplied as the parameter, then the extend processor attaches the extent file to the database. As with create, the extent file must have been previously allocated. There is a maximum number of extents which can be attached to a database, and an error message will result if this limit is reached. If the database was created with no vocabulary extent, the extend command can be used to attach one. In this case, the vocabulary block size will be requested.

If the "directory" keyword is supplied as the parameter, the document and vocabulary directories are extended by one physical block each, making the database directory file two blocks longer. If the directories already contain 4096 entries, they cannot be enlarged and an error results.

4.2.3 Detach

The detach command removes the specified text or vocabulary extent from the database, and has the following forms:

```
database/DEtach:TEXT number
database/DEtach:VOCabulary number
```

The extent number specifies the extent to be detached, but this must be the last extent of its type. The extents are numbered starting with 1. Also, the extent must be empty (contain no document or vocabulary files). For example, if there are three text extents, only number 3 can be deleted, and only if it is empty. If the last extent is not empty but deletion is desired, its files it can be moved to another extent (assuming there is enough space) using the move command. After an extent has been detached from a database, the file may be deleted with PIP.

4.2.4 Zero

The zero command provides a shortcut method for zapping all the data stored in a database. Upon completion, the database is the same size, but empty, as if it had just been created. Its format is simply:

```
database/ZEro
```

The document and vocabulary directories are cleared, the bitmaps are reset, and new data block sizes are requested. The data block sizes must conform to the rules given under the create command.

4.3 Document and Vocabulary Updating

The major function of EURUP is to provide database updating capabilities, and it does so by operating on quanta of documents. To update a database, it must first be opened (see Section 4.5.1). Update mode commands then insert or replace documents from input text files, each file containing one complete document (see Section 3.0). Once a document has been inserted, a vocabulary for that document can be generated. If desired, whole documents or vocabularies can be deleted. If a document is deleted or replaced, the associated vocabulary file is always deleted.

Since the updating features will be used primarily for constructing databases, adding new documents and occasional maintenance, there is little need for "incremental" updates, that is, the ability to edit a database at the word level. For such a capability to work nicely, a more or less complete, interactive text editor would have to be implemented. With this, the document text, vocabulary file and database index would be updated at the time of an edit. The cost of such a feature would be ridiculous when compared with its usefulness. If text in a database is found to be in error, the document input file can be edited using a standard text editor. Then that whole document can simply be replaced in the database. This procedure works well even if only a single word is changed, for replacement is "smarter" than a document deletion followed by insertion.

For each document or vocabulary processed, statistics are written to a log file, which defaults to the terminal. This includes the total time, the number of words in the document and the number of unique terms. Because a batch of documents takes a long time, the document numbers and processing time are always logged at the terminal in order to provide a progress report.

4.3.1 Insert

The insert command inserts documents or vocabularies into the open database and is the work-horse of database construction. A single document or a range of documents can be processed with one command of the following forms:

```
INSert DOCuments number[..number] [,log file[/DE]][<input specifier]
INSert VOCabularies number[..number] [,log file[/DE]]
```

The number or number range specifies the documents or vocabularies to be inserted. The log file and/or input specifier are in the form of a standard DOS CSI string, and either can be null. The log file is normally opened for extension, but can be deleted first by adding the "/DE" switch.

In the case of inserting documents, the document input files must have names which begin with a two (or one) letter prefix followed by a four (or five) digit decimal document number (document 0 is allowed). The prefix letter(s) can be supplied with the input specifier. The number of the document being processed is concatenated with this to form the complete file name. The input file specifier can also identify the device, unit, file name extension and user code. The defaults for each of the fields of the input specifier are:

device and unit	SY:
file name	the first two characters of the database name concatenated with a 4 digit document number
file name extension	DOC
user code	the current user code, not [1,5] as with the database extents

Examples that should clarify the use of this command are given in Section 5.

When inserting vocabularies, no input is required, for the document is obtained from the database itself. Obviously, the documents must have been previously inserted, or an error results. In addition, a vocabulary extent must exist, and term numbers must have been assigned (see Section 4.5.7 for details of term number assignment).

4.3.2 Replace

The replace command is similar to the insert command in syntax and function. Its two forms are:

```
REPlace DOCuments number[..number] [,log file[/DE]][<input specifier]
REPlace VOCabularies number[..number] [,log file[/DE]]
```

The only differences between the replace commands and the insert commands are that: 1) No error will result if a given document or vocabulary already exists in the database. It will merely be overwritten. 2) If a document is replaced, its corresponding vocabulary is deleted if it exists. Note that replacing a non-existent document or vocabulary is equivalent to inserting it.

4.3.3 Delete

The delete command has two simple forms, namely:

```
DELete DOCuments number[..number] [,log file[/DE]]
DELete VOCabularies number[..number] [,log file[/DE]]
```

These commands merely delete whole documents or vocabularies from the database. When documents are deleted, their associated vocabulary files are also deleted. When vocabularies are deleted, the document text in the database is unaffected.

4.4 Extent Reorganization and Garbage Collection

A secondary purpose of EURUP is to provide procedures for reorganizing a database. As the database is modified by adding and removing documents, the physical organization of the data becomes disheveled. The index in particular, which is stored as many sorted, linked lists, becomes very jumbled. As items are inserted, blocks of terms or postings often have to be physically split and moved. In addition to this movement, which usually causes longer access times, small holes of unused space get generated. These holes either don't or can't be reclaimed by the normal updating procedures. Oftentimes the updating procedures will not be able to find usable free space in the index, even though it is only half full. In other words, half the index extent has become garbage. A similar problem exists for text extents, but it is not nearly as pronounced.

Reorganization restores the data to a "clean" physical arrangement and collects all the garbage into available free space. Separate procedures are available for reorganizing each kind of extent in a database. The index can be reorganized without requiring access to any text or vocabulary extents. Similarly, each text extent can be reorganized independently, as can the vocabulary extents.

The process of reorganizing an extent is fairly straightforward. Data is copied in a logically ordered way to a simple sequential file, the extent is reset to be empty, and then the data is copied back. For example, the index is done by writing a term, followed by its associated postings file. The terms and their postings files are written in lexicographic order. When this sequential file is copied back, each postings file winds up physically adjacent to its term, which just happens to be optimal for a moving arm disk.

The reorganization procedures are broken up into two phases, for reasons that will be made evident in Section 5. The first phase (the dump command) does a structured copy of data from a database extent to a sequential file. This results in no change to the database. The second phase (the load command) zeroes an extent and copies the sequential form back into it. A single convenient command is available to do both phases, using a temporary file.

Since reorganization is done on an extent basis, there doesn't need to be enough on-line storage to hold the whole database as well as space for the sequential form of an extent. Only the database directory, the extent to be reorganized and enough free space to hold a sequential form of the extent need to be on-line. This means that a database can be created which uses all the on-line storage and it still can be reorganized (assuming the system has more than one disk drive).

4.4.1 Dump

The dump command copies an extent to a sequential file, having one form for each extent type.

```
database/DUmp:INDEX,file
database/DUmp:TEXT number,file
database/DUmp:VOCabulary number,file
```

For the first form, the database index, which is always a single extent, is dumped. For the other forms, "number" specifies which of the text or vocabulary extents is to be dumped.

The given "file" is created by the dump command and must not exist. It can be created in any user code, being put in [1,5] by default. The default file name extension depends on the extent type, being "LX" for the index, "LT" for text and "LV" for vocabulary extents.

Note that there typically must be a large amount of free space for the dump file, for it may be as large as the extent file. If the device runs out of space during the dump, it is aborted and the file is deleted. An error message is logged which attempts to give some indication of how far the dump had progressed and how much data was left to be dumped.

4.4.2 Load

The load command resets an extent and copies a previously dumped linear form back into the extent. The command forms are:

```
database/LOad:INDEX<file
database/LOad:TEXT number<file
database/LOad:VOCabulary number<file
```

The load processor first finds and verifies the contents of the specified input file. The same defaults apply to the file that are used in the dump command. Assuming no error arises, all data is removed from the specified extent and its bitmap is reset. At this point, the data block size is requested. For the index, both the terms and postings block sizes can be entered. If it is desired to leave the block size what it was, simply type return. Otherwise, the entry must be a multiple of the physical block size, as in the create and zero commands. Finally, data from the input file is copied into the extent. The input file is not deleted.

4.4.3 Reorganize

The reorganize command is essentially shorthand for a dump command followed by a load command. It has three forms similar to the dump command.

```
database/REorganize:INDEX [,temp file]
database/REorganize:TEXT number [,temp file]
database/REorganize:VOCabulary number [,temp file]
```

The descriptions given with the dump and load commands apply to the reorganize command except that a temporary file is used instead. Since the system device, "SY:", usually doesn't have much free space, the temp file can be specified explicitly. Normally, only a device and/or user code is needed to specify which directory to put the temporary file in. The default temporary file name depends on the extent type, being "INDEX", "TEXT" or "VOCAB", with an extension of "TMP". It is put in user code [1,5] by default. The temporary file is deleted if it exists before doing the reorganization, and is deleted upon completion. As with load, new data block sizes can be specified.

4.5 Miscellaneous

4.5.1 Open

The open (or update) command opens a database and sends EURUP into update mode. It has two equivalent forms:

```
database[/OPen]
database[/UPdate]
```

Opening a database involves locating the various database extents. Once the database is open, EURUP enters update mode and prompts the user with a "*". Update mode commands can then be given to modify the database. Notice that open is the default command. If no command switch is specified in command mode, open is assumed.

4.5.2 Exit

There are three alternative commands which can be used to exit update mode. These are:

```
EXit
CLose
BYe
```

The currently open database is closed, and EURUP returns to command mode, prompting the user with another "#".

4.5.3 Directory

The directory command lists useful information contained in the database directory. Its format is:

```
database/DIrectory[,log file]
```

The primary purpose of the directory command is to print the extent directory, showing the file specifications for the extents in the database. It also shows which documents and vocabularies are in the database, as well as which extents they are contained in.

4.5.4 Change

The change command is used to change any of the symbolic extent pointers stored in the database directory. Its various forms are:

```
database/CHange:INDEX<file
database/CHange:TEXT number<file
database/CHange:VOCabulary number<file
```

The change command processor overwrites the specified extent pointer with the input file specification. In the case of a text or vocabulary extent, the database must actually contain the specified extent. No user code can be given with the file, for all extents are assumed to be in [1,5]. The specified file is not looked up. Only the database directory needs to be on-line.

This command allows the user to manually control where the extents of a database are located and is needed because the operating system does not have a volume naming capability. This control was motivated by the expectation that database directories (as well as other extents) will be copied from disk to disk, so that different sets of databases can be on-line simultaneously.

As an example of the use of this command, assume that a database consists of a text extent on one volume, and that the database index and database directory are on the system volume. If it desired to move the index extent from the system volume to the volume with the text, it can simply be moved using PIP. Now the change command can be used to fix the index extent pointer in the database directory, setting it to point to the new file on the other volume. Finally, PIP can be used to delete the original copy of the index from the system volume.

When copying extents to other volumes and using the change command, care must be taken to maintain a "master" copy of the database. If separate updates are done to two different copies of a database which have some extents in common, both databases will be ruined. For example, if there are two copies of a database directory (and hence the document directory) and they both refer to the same text extents, the replacement of two documents, one using each directory, will result in both directories being incorrect, with no easy way to find and correct the problem.

4.5.5 Select

The select command specifies which text or vocabulary extent document or vocabulary files are inserted in. It has the following two forms:

```
        SELect TEXT number
        SELect VOCabulary number
```

When update mode is entered, the first text and vocabulary extents are selected. If either of these is already full, the user must select another extent (assuming one is available). EURUP does not automatically select a relatively empty extent. This command affects the operation of the insert, replace and move commands.

4.5.6 Move

The move command moves document or vocabulary files from one extent to another. Its two forms are:

```
    MOVE DOCuments number[..number]
    MOVE VOCabularies number[..number]
```

The specified documents or vocabularies are moved from their present locations to the currently selected text or vocabulary extent. No operation results if they are already in the selected extent. This command does not change the contents or logical structure of the database. Only the physical organization of the data is affected.

4.5.7 Numbers

As mentioned in Sections 2.1 and 2.3, identifying numbers are assigned to index terms for use in the vocabulary files. The assignment of term numbers is optional and is controlled by the numbers command, which has two forms.

```
NUMBERS ON
NUMBERS OFF
```

When a database is created, the term numbers option is off, and no term numbers are assigned. Entering the "numbers on" command causes all new numbers to be assigned to the index terms. It was mentioned in Part 1, Section 2.9, that the EUREKA freq command prints out three numbers with each term, the first two being the document count and frequency. The third number printed by the freq command is the low order 16 bits of the term number in octal (remember that the first character of the term forms the high 8 bits of the number).

If the term numbering option is on, then any new terms that get entered in the index (via the insert or replace commands) are assigned a number. To retain proper ordering, this number must fall between the numbers assigned to adjacent terms. If the terms surrounding the new term have consecutive numbers, then a reassignment must be done to one of the old terms in order to accommodate the new term. This procedure requires considerable time and is the reason why term numbers are optional. If the term numbering option is off, new terms are inserted into the index without being assigned a number.

When constructing a database by inserting documents, term number conflicts start occurring very soon and frequently. Hence it is wise to construct a database with the numbering option off. Since vocabularies cannot be generated until numbers have been assigned, the vocabularies cannot be built as the documents are being inserted. It turns out that this is okay because the word stemming done during vocabulary generation works much better if the index is complete.

Once the index has gotten fairly large, it becomes relatively stable and numbers can be turned on. More documents can be added without term number conflicts, but if many documents get inserted, reassignments will start occurring. Any reassignments that take place cause a message to be logged, and if they become significantly frequent, numbers can be turned off again.

Since the vocabulary files use term numbers, any time the term numbers are regenerated, or when the option is turned off, all vocabulary files are automatically deleted. As a large existing database is updated, occasional term number reassignments should be tolerated. The alternative is to turn off the numbering option, which deletes all vocabulary files, and rebuild the vocabularies after the updates are done.

4.5.8 Stop

The stop command constructs the database stop and save lists for use during vocabulary building. One command form converts the stop and save lists into compact, coded files. The other is an aid to generating the stop list itself.

```
STOP
STOP letter [automatic document bound, auto frequency bound, lower
            document bound, lower frequency bound]
```

In the first form, four files are involved, each with the same file name as the database. The stop and save list inputs were discussed in Section 2.5, but the resulting compacted files were not. The outputs are "database name.STP[1,5]" and "database name.SAL[1,5]" for the stop and save parts, respectively. The inputs are mapped into a series of records, one per letter. The records are sorted alphabetically, and the words within each record are sorted. Each input word becomes two PDP-11 bytes, representing the low 16 bits of its type number. The list for a letter terminates with a -1. A dummy entry, produced when a letter has no entries (e.g. "X"), has only the -1. There are up to 26 records, with one for each letter up to the last one specified in the input. Thus, if no

"Z" words are on the stop list, but some for "Y" are, then the 25 records for "A" to "Y" are built, even if some are empty. This permits knowing the current letter by simple counting.

The text files may be constructed entirely by hand with the editor. For the extended stop command, the letter must appear, but the numbers are optional. The stop list text gets rebuilt starting at the given letter, using the four numeric parameters (or defaults). Suppose the letter given was "C". Any existing file lines for "A" and "B" are not altered, but those for "C" are replaced. After the letter is finished, the user is asked if he wants to continue with the next letter. A "no" response causes an exit with the remaining letters unmodified.

The command scans the database index and picks stop list candidates, using the four parameters (the current defaults being 150,25,20 and 5). Document counts and maximum frequencies are collected for each root form (after stemming) and compared to the bounds. If the values surpass either the automatic document count or automatic frequency value, the word is displayed and added to the stop list text. If both values fall below the respective lower bounds, the word is skipped. For those in between, the word is displayed and the user gets a prompt. A "y" response adds the candidate to the list and anything else skips it. Thus the user builds the list with help from the machine.

The text file may be edited later to make any changes caused by rethinking decisions about words. Terms may be added or deleted easily.

The additional file "database name.TMP" is produced during the stemming phase. It is a list of original and reduced forms of words that were stemmed successfully. It is useful for monitoring the stemmer for possible modifications and for catching some save list candidates. That is, some words in the temporary file may have to be put on the save list to prevent incorrect or undesired stemming.

4.5.9 Unlock

The unlock command is an update mode command which recovers a locked database. It is entered with the single keyword "UNLOCK". If the system (EURUP, the operating system, the hardware or power) goes down while a database is being updated, chances are that it will be left in a locked state. When a database is locked, the updating and reorganizing commands will not function, but will issue a message to the effect. When this happens, the database can usually be restored by opening and unlocking it.

5. Comments, Suggestions and Examples on the Use of EURUP

5.0 Some Database Characteristics

It might be instructive to first give some facts and figures about various database characteristics and what works well with EURUP and EUREKA. Since our system uses 512-byte physical disk block sizes exclusively, all references to blocks will mean 512-byte blocks. A disk pack holds 48,720 blocks, so any single extent is limited to about 48,000 blocks.

As mentioned before, the number of documents in a database is limited to 4,096. Typically, documents range anywhere from 100 to over 10,000 words. There is really no physical limit to the size of a document, but EUREKA has practical limits, in that full-text searching response time is proportional to the document size. The average word length is almost always between 5 and 6, so a block holds about 75 - 80 words (taking spaces into account). An average document size of about 2,000 words (or about 25 blocks) works very well for EUREKA, as does anything smaller. The system begins to operate noticably slower on documents over 8,000 words (around 100 blocks).

Other interesting characteristics of textual databases are the token to term ratio (i.e. the number of words to the number of unique words), the total number of terms and the total number of postings. The token to term ratio varies widely with document size and subject material, but it typically falls between 2 and 10 at the document level. The total number of terms in a database depends on its size and subject matter. A highly specific database like legal text will have fewer terms than a general one of the same size. The total number of postings is highly variable, but is related to the token to term ratio, average document size and total number of documents.

To demonstrate the variability of database characteristics and the amount of data that EUREKA handles, some statistics are given for two medium sized databases.

database	# text blocks	# docs	# terms	# postings
specific	42,000	3,200	21,000	683,000
general	12,000	500	43,000	375,000

5.1 Constructing an Average Database

This section gives some examples on the use of various EURUP commands. The complete procedure of building a database is stepped through. In order to give a feel for how EURUP is used, various problems that may arise are discussed and the way such problems might be handled are outlined.

The first thing to do is decide how much space should be allocated for the various extents of the database, based on the size of the raw input text. At this point, only a rough approximation can be made, but it will have to do. You don't have to sweat it though, because extents can be made larger or smaller, or new extents can be allocated and attached.

A good approximation for the text extent is to make it the same size as the raw input text. The index is much more of a problem, because the amount of space needed is dependent on a variety of factors, such as the number of documents, average document size, and the token to term ratio. Most of our existing databases have index extents which are less than 20% the size of the text. However, due to the disorganization which arises while inserting many documents, it is better to have an oversized index. During construction the index gets full long before the database is complete. However, lost space can be recovered by reorganizing the index. Document insertions can then continue until the index has to be reorganized again. After the database is complete, the index extent can be shrunk to the size needed. A good rule of thumb is to allocate an index extent which is maybe 30-40% the size of the text. If vocabularies are going to be generated, a vocabulary extent of around 25% the size of the text is a reasonable starting size.

Now let's get down to a concrete example. Assume we want to make a database from 24,000 blocks of raw text stored on a disk pack, and that we have a formatting program which will break it up into suitably sized documents, say 2,000 documents of about 1,000 words each. The first thing to do is find a pack with enough space to put the database on, then get into user code [1,5] and use PIP to allocate files. It goes something like this:

```
.LO 1,5
.P
PIP  V10-02
#DR1:/EN                           (ensure user code [1,5] exists)
#DR1:SAMPLE.NDX/AL:10000           (for the index)
#DR1:SAMPLE.TXT/AL:24000           (for the text)
#DR1:SAMPLE.VOC/AL:6000            (for vocabularies)
#^C
.KI
.
```

The next thing to do is use EURUP to create the database. Note that the create command prompts the user for the directory size and data block sizes. EURUP runs under EUREKA's executive "exec". The code library for EURUP is "EURUP.CIL[13,21]", which must be assigned to "CIL" in order to make exec run EURUP rather than EUREKA.

```
.LO EURUP                       (or LO 13,21)
.AS EURUP.CIL,CIL
.EXEC
EURUP - Eureka Update  V02
#SAMPLE/CR<SAMPLE.NDX,SAMPLE.TXT,SAMPLE.VOC
Max document number? 2000
Terms blocksize (bytes)? 1024
Postings blocksize? 512
Text blocksize? 3072
Vocabulary blocksize? 2048
#^C
.KI
.
```

The result of all this is the creation of a database named "sample". File "SAMPLE.DB[1,5]" is allocated on the system volume as the database directory. Note that the specified block sizes must be a multiple of the physical block size, which is 512. The numbers given in the example are more or less standard, but all except the postings block size can be varied if desired. The postings block size <u>must</u> be 512 because of restrictions in EUREKA.

The next thing to do is format a few documents and insert them into the database. A small fraction of the database is thus created in order to check out the formatting program. For our example, this involves demounting the pack containing the database and mounting the pack with the raw text. Let's assume that 20 documents are formatted into files "SD0001.DOC" thru "SD0020.DOC" in user code [30,1] on the system volume. Next, the database pack is remounted, and the following is done:

```
(in EUREKA command mode)
#SAMPLE                                      (open database "sample")
*INSERT DOCS 1..20,SAMPLE.LOG<SD[30,1]       (insert the 20 docs)
*EXIT
#
```

For the assumed document size, the insert command will probably run for 15 to 20 minutes (it should take less than 1 minute per document). Statistics and error messages are sent to file "SAMPLE.LOG". This file should be checked to see if any invalid markup codes or characters were found in the input. Statistics for each document logged to this file include the number of tokens, number of terms, number of new terms added to the index, total time and the like. Assuming all went well, EUREKA can now be used to examine the 20 documents and see how they turned out, with a command like "P FROM [1,2,3,4,5]". If EURUP found errors in the document input format, or if the structure of the documents is not correct, it is undoubtedly a fault of the formatting program (but may be the raw text itself). After fixing the problem, EURUP can be given a command to zero "sample", and the documents can be inserted again.

Once the formatter is satisfactory, it is probably a good idea to try generating a few vocabularies and check them using the words command in EUREKA. This is easily done with:

```
(in EURUP command mode)
#SAMPLE
*NUMBERS ON                 (make term number assignments)
*INS VOC 1..20              (generate vocabularies, log to terminal)
*EXIT
#
```

The term number assignments can be checked with the EUREKA freq command.

Now we are ready to build the whole database. It is a good idea to do batches of say 200 to 400 documents rather than using a single insert command, because the system will run for so long. Besides, the documents are being formatted from the raw text pack to the system pack, and there might not be enough space to hold them all at one time. The term numbering option is turned off first. After a batch, or if the index becomes full due to disorganization, the index is reorganized. Between batches, the raw text pack has to be mounted in order to format the next batch of documents. A single batch is handled by the following commands:

```
(in EURUP command mode)
#SAMPLE
*NUMBERS OFF                          (only needed once)
*INS DOC 21..400,SAMPLE.LOG<SD[30,1]  (then 401..800, etc)
*EX
#SAMPLE/RE:INDEX,DR1:                 (reorganize the index)
```

Note that the remaining space on the database pack is specified for use by the large temporary file needed to reorganize the index. On the insert command, "SAMPLE.LOG" is extended, but it could be deleted at the beginning of the first batch by adding the "/DE" switch. If the index really becomes full and no documents can be inserted after reorganizing it, then it must be enlarged. The following example shows how this can be done.

```
.AS EURUP.CIL[13,21],CIL
.EXEC
#SAMPLE/DU:INDEX,JUNK        (dump the index to SY:)
#^C
.KI
.LO 1,5
.P
#SAMPLE.NDX/DE               (delete the small file)
#SAMPLE.NDX/AL:12000         (make it bigger, same name)
#^C
.KI
.EXEC
#SAMPLE/LO:INDEX<JUNK        (reload the index)
#^C
.KI
.P                           (delete the index dump file)
#JUNK.LX[1,5]/DE
#^C
.KI
.
```

This example does not really take into account the fact that the needed space may not be available or contiguous, but at least the general idea of the procedure is demonstrated. The same kind of procedure can be used on text or vocabulary extents too. The extent could be made smaller rather than larger and it could be moved to another volume at the same time as well.

After all the documents have been inserted, term numbers can be assigned and the vocabularies generated. These procedures will go noticably faster if the index is reorganized before doing them. Stop and save lists should be created before inserting the vocabularies (see Section 4.5.8), but the stop command must be given after term numbers have been assigned in order to encode the lists. Assuming the textual versions of the stop and save lists have been made, vocabularies are generated thus:

```
(in EURUP command mode)
#SAMPLE/RE:INDEX,DR1:
#SAMPLE
*NUMBERS ON
*STOP                       (encode the stop & save lists)
*INS VOC 1..2000            (you might use batches here too)
*EX                         (the database is complete!)
#
```

It is likely that after using vocabularies for awhile, users will want changes made to the stop and save lists. In that case the files should be edited, another stop command done and then the vocabularies can be rebuilt with the replace command.

One more example is given to clarify the use of the change command. Using the sample database from above, the idea is to make a system on the same pack as the text and index, and move the database directory to that pack too. Once this is done, the pack can be mounted on unit 0, the system booted up and EUREKA used to search "sample". The following sequence should do the trick.

```
.LO 1,1                      (copy across needed system files)
.P
#DR1:/EN
#DR1:<MONLIB.CIL             (these are the bare necessities)
#DR1:<P,EXEC.LDA             (others may be desired)
#DR1:<EUREKA.CIL[13,13]      (EUREKA scratch & user files
#^C                           will also be needed)
.KI
.WRBOOT                      (put a bootstrap on the pack)
#DR1:
A120 000001
.CO
.AS EURUP.CIL[13,21],CIL
.EXEC                            (use change command)
#SAMPLE/CH:INDEX<SAMPLE.NDX      (db will be on SY:)
#SAMPLE/CH:TEXT1<SAMPLE.TXT      (file names don't change)
#SAMPLE/CH:VOC1<SAMPLE.VOC
#^C
.KI
.LO 1,5
.P                               (copy across database files)
#DR1:<SAMPLE.DB,SAMPLE.CTX       (assuming .CTX was made)
```

5.2 Using EURUP and EUREKA to Check Spelling

It turns out to be fairly easy to use EURUP and EUREKA to check all the terms used in a manuscript being prepared for printing on-line. This is an easy way to catch spelling errors and typos. The idea is to create a database containing one document, namely the manuscript, and then use the freq or term command in EUREKA to scan the whole index. If an error is detected in the index, then that term can be searched for and printed in order to find the location of the error within the text.

The easiest way to go about this is to begin with a version of the manuscript which is in print format, with no funny markup codes. Four things must be done with this file.

1) It must be given a name like "XX0000.DOC".

2) Asterisks must be doubled up so that they will remain as text.

3) A section code, like "*C1", must be inserted at the front of the file.

4) Paragraph and or sentence codes should be sprinkled into the text. This will make it easy to locate any errors that are found, using a "P S" or "P P" command in EUREKA.

The following simple edit commands will make these changes.

```
.E
#XX0000.DOC<PAPER.SET
*10000HT/*/**/
*EX
#XX0000.DOC
*I/*C1/
*10000HT/                         /*P/
*EX
#XX0000.DOC
*10000HT/     /*S/
*EX
#^C
.KI
.
```

Changing long strings of spaces to "*P" is intended to put a paragraph mark in front of each page number. The short string of spaces should be the amount used to indent paragraphs, the result being that a sentence mark is put in front of each paragraph. Other paragraph and sentence marks invariably trickle in, but they don't pose any problems. You could try inserting a sentence mark after each period instead, or do just about anything else you like.

Once the manuscript is in standard document format, the database can be trivially constructed as follows:

```
.LO 1,5                  (first alloc the files)
.P
#SPELL.TXT/AL:200        (manuscript size + breather space)
#SPELL.NDX/AL:200        (same size as text works okay)
#^C
.KI
.AS EURUP.CIL[13,21],CIL       (run EURUP)
.EXEC
#SPELL/CR<SPELL.NDX,SPELL.TXT
Max document number? 1
Terms blocksize (bytes)? 512
Postings blocksize? 512
Text blocksize? 2048
#SPELL
*INS DOC 0 < XX[your user code]
*EX
#^C
.KI
.
```

Now you can fire up EUREKA and have at it!

RELATED REFERENCES

Each paper listed below discusses some aspects of EUREKA and EURUP. They should be consulted for more details of how the systems work. All theses are from the Department of Computer Science, University of Illinois at Urbana-Champaign.

Bahar, M., "Implemention of an Information-retrieval Based CAI System," M.S. thesis, Dept. of Computer Science Report No. 77-902, Aug. 1977.

Burket, T. G., "User Aids and Document Clustering with Word Frequencies," M.S. thesis, Feb. 1979.

Emrath, P. A., "EUREKA Update," M.S. thesis, Oct. 1977.

Leung, W. M., "A Unique Word-scanning Facility for the EUREKA Full-text Information Retrieval System," M.S. thesis, Dept. of Computer Science Report No. 78-914, Jan. 1978.

Milner, J. M., "A Multiprocess, Multiuser Executive for an Experimental Information Retrieval System," M.S. thesis, Dept. of Computer Science Report No. 75-736, Aug. 1975.

Morgan, J. K., "Description of an Experimental On-line, Minicomputer-based Information Retrieval System," M.S thesis, Dept. of Computer Science Report No. 76-779, Feb. 1976.

Morgan, T. J., "A Thesaurus Feature for the EUREKA Information Retrieval System," M.S. thesis, Dept. of Computer Science Report No. 77-855, May 1977.

Rinewalt, J. R., "Evaluation of Selected Features of the EUREKA Full-text Information Retrieval System," Ph.D thesis, Dept. of Computer Science Report No. 76-823, Sept. 1976.

Stellhorn, W. H., "An Experimental Information Retrieval System," Dept. of Computer Science Report No. 74-657, July 1974.

IBLIOGRAPHIC DATA HEET	1. Report No. UIUCDCS-R-79-956	2.	3. Recipient's Accession No.

Title and Subtitle

USER'S GUIDE TO EUREKA AND EURUP

5. Report Date February 1979

6.

Author(s)

Thomas G. Burket and Perry A. Emrath

8. Performing Organization Rept. No. UIUCDCS-R-79-956

Performing Organization Name and Address

University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

US NSF MCS77-27910

. Sponsoring Organization Name and Address

National Science Foundation
Washington, D. C.

13. Type of Report & Period Covered

Technical Report

14.

Supplementary Notes

. Abstracts

EUREKA is an experimental system designed for research in the field of information retrieval. Part 1 of this report describes the current state of EUREKA and how to use it to conduct searches of a database. It is divided into four sections--basic features, query language description, general comments about EUREKA use and a glossary of terms and rules.

EURUP, for EUREKA UPDATE, is the system which builds and maintains databases for use by EUREKA. Part 2 discusses the logical organization of a database, as well as how to construct or modify the component files. It has five sections--introduction, database structure, EURUP input and output formats, EURUP functions and command language, and general comments about efficient EURUP use. Both EURUP and EUREKA run on a PDP-11/40 minicomputer.

Key Words and Document Analysis. 17a. Descriptors

Database organization and maintenance
Full-text document retrieval
Inverted file
On-line information retrieval
User manual
User search aids

. Identifiers/Open-Ended Terms

. COSATI Field/Group

Availability Statement RELEASE UNLIMITED	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 99
	20. Security Class (This Page UNCLASSIFIED	22. Price